

ALIGN.CPP

7

```

    *p1 = *p2;
    *p2 = ftmp;
    p1++, p2++;
}

offset = dim2*dim;
for(i=0; i<dim2; i++){
    p1 = array[dim2*i*dim];
    p2 = array[offset+i*dim];
    for(j=0; j<dim2; j++){
        ftmp = *p1;
        *p1 = *p2;
        *p2 = ftmp;
        p1++, p2++;
    }
}

return(0);
}

int convert_to_magnitude(
float *out,
float *in,
int dim
){
    int i, j, dim2 = dim/2;
    float *preal, *pimag, *pout, *ftmp;

    preal = in;
    pimag = in[dim];
    pout = out;
    for(i=0; i<(1+dim2); i++){
        for(j=0; j<dim; j++){
            ftmp = *preal * *preal + *pimag * *pimag;
            *pout = (float)sqrt( (double)ftmp );
            preal++; pimag++; pout++;
        }
        preal=dim;
        pimag=dim;
    }

    return(1);
}

int convert_to_magnitude_id_inplace(
float *real,
float *imaginary,
int dim
){
    int i, dim2 = dim/2;
    float *preal, *pimag, *ftmp;

    preal = real;
    pimag = imaginary;
    for(i=0; i<dim; i++){
        ftmp = *preal * *preal + *pimag * *pimag;
        *(preal++) = (float)sqrt( (double)ftmp );
        pimag++;
    }

    return(1);
}

int log_polar_remap(
float *in,
float *out,
int dim
){
    int i, dim2 = dim/2, xx, yy, j, jj, k;
    float *pin, *pout, *ftmp, *radius[MAX_LINEAR_DIMENSION];
    double theta, dx, dy, radius[MAX_LINEAR_DIMENSION], x, y, fracc, fracy, *pradius;
    scale_increment=pow( 1.0/(double)START_RADIUS, 1.0/(double)lp_sampling );
    for(i=0; i<lp_sampling; i++){
        radius[i] = (START_RADIUS*(double)dim2) * pow(scale_increment, (double)i);
    }

    pout = out;
    for(theta=0.0; j=0; j<lp_sampling; j++, theta += (PI/lp_sampling)){
        dx = cos(theta);
        dy = sin(theta);
        pradius = radius;
        pout = &out[j];
        for(i=0; i<lp_sampling; i++){

```

```

x = (double)dim2 * pradius * dx;
y = (double)dim2 * pradius * dy;
xx = (int)x;
yy = (int)y;
fracy = x - (double)xx;
fracy = y - (double)yy;
pin = &inxy*dim + xx;
*point = (float) ( (1.0-fracy)*(1.0-fracy)*(double)*(pin++) );
*point += (float) ( fracy*(1.0-fracy)*(double)*pin );
pin += (dim-1);
*point += (float) ( (1.0-fracy)*fracy*(double)*(pin++) );
*point += (float) ( fracy*fracy*(double)*pin );
pout += lp_sampling;
}

/* now filter it along the scale axis */
for(i=0;i<lp_sampling;i++){
    pout = ftemp;
    for(j=0;j<lp_sampling;j++){
        *point = (float)0.0;
        for(k=- (LOG_MOV_AVG/2); k<= (LOG_MOV_AVG/2); k++){
            if(jj<0)jj=0;
            else if(jj>= lp_sampling)jj=lp_sampling-1;
            *point += out[i+jj*lp_sampling];
        }
        *point += (float)LOG_MOV_AVG;
    }
    pin = ftemp;
    pout = kout[i];
    for(j=0;j<lp_sampling;j++){
        *pout += *pin++;
        pout += lp_sampling;
    }
    pout = ftemp;
    for(k=- (LOG_SMOOTH/2); k<= (LOG_SMOOTH/2); k++){
        jj=j+k;
        if(jj<0)jj=0;
        else if(jj>= lp_sampling)jj=lp_sampling-1;
        *pout += out[i+jj*lp_sampling];
    }
    *pout += (float)LOG_SMOOTH;
}
memcpy(kout[i],ftemp,lp_sampling*sizeof(float));
return(i);
}

float get_median(float(float *median){
    if(median[0] > median[2]) return( -median[0] - median[2])/(median[1] + median[0] - 2*median[2]);
    else return( median[2] - median[0])/(median[1] + median[2] - 2*median[0]);
}

/* this is the fft window profile for mitigating edge effects; change to other windows if their better
or..., maybe certain windows are better for certain tasks, e.g., log polar vs. straight correlation
int load_windowing_function(int dim,float *window){
    int i;
    double step,x,y;
    step = 2.0*PI / (double)(dim+1);
    for(i=0,x=step;i<dim;i++,x+=step){
        y = (1.0 - cos(x))/2.0;
        window[i] = (float)sqrt(y);
    }
    return(i);
}

int window_id_vector(
    float *array,
    int data_length,
    int full_length
){
    int i;
    float *parray,*pwindow;
    float *window_function = new float(data_length);
    load_windowing_function(data_length,window_function);
    parray = array;

```

```

dott = (float)1.0 - dott*dott;
if (dott<(float)0.0) dott=(float)0.0;
dott = (float)sqrt((double)dott);
cross = *preall + *pimaginary2++ - *(preal2++) * *pimaginary1;
if (cross < (float)0.0) cross = -(float)1.0;
else cross = (float)1.0;
ftmp = mag2;
dott*=ftmp;dott*=ftmp;
*preal1++ = dott;
* (pimaginary1++) = cross*dott;
}
preal1+=dim;
pimaginary1+=dim;
preal2+=dim;
pimaginary2+=dim;
}

/* now back into the original domain, then shift the array for simplicity */
realfft2d_in_place(real1.bits,1,wr.w1;
shift_array(real1,dim);

/* then find the top 'candidate' number of points, loading their parameters along the way */
for(i=0;i<number_candidates;i++){
highest = -(float)1e20;
preall = real1;
for(j=0;j<dim;j++){
if (preall > highest){
/* check to see if this is within PICK_RADIUS of a previous choice */
ok = 1;
l = i;
while( l-- > 0 ){
if( abs(j-y_off[l]) < PICK_RADIUS ||
abs(j-dim-y_off[l]) < PICK_RADIUS ||
abs(j-dim-y_off[l]) < PICK_RADIUS ){
if( abs(k-x_off[l]) < PICK_RADIUS ||
abs(k+dim-x_off[l]) < PICK_RADIUS ||
abs(k-dim-x_off[l]) < PICK_RADIUS ){ok=0;
}
}
if(ok){
highest = *preall;
x_off[i] = k;
y_off[i] = j;
}
preall++;
}
}

pin = in;
memset(out,0,outdim*outdim*sizeof(float));
for(i=0;i<ydim;i++){
pout = &out[i*outdim] * outdim;
for(j=0;j<xdim;j++){
pout[j/downsample] += (float)*(pin++);
}
}

/* normalize it for downsampling
if (downsample > 1){
xdim = 1 + (xdim-1)/downsample;
ydim = 1 + (ydim-1)/downsample;
normalize = (float)downsample * (float)downsample;
for(i=0;i<ydim;i++){
pout = &out[i*outdim];
for(j=0;j<xdim;j++){
*(pout++) /= normalize;
}
}
}

if(WINDOW_ORIGINALS){
float *window_function = new float[outdim];
load_windowing_function(xdim>window_function);
pout = out;
for(i=0;i<ydim;i++){
pwindow = window_function;
for(j=0;j<xdim;j++){
*(pout++) *= *pwindow++;
}
pout+=(outdim-xdim);
}
load_windowing_function(ydim>window_function);
pout = out;
for(i=0;i<ydim;i++){
pwindow = &window_function[i];
for(j=0;j<xdim;j++){
*(pout++) *= *pwindow;
}
pout+=(outdim-xdim);
}
delete [] window_function;
}
return(l);
}

```

```

int fourier_mellin_transform(
    float *in,
    float *out,
    int dim,
    float *ftemp,
    float *out)
{
    int i,j;
    float *pout,*pwindow;

    convert_to_magnitude(ftemp,in,dim);
    log_polar_remap(ftemp,out,dim);
    if(WINDOW_LOGPOLAR_LOG){
        float *window_function = new float[lp_sampling];
        load_window_function(lp_sampling,window_function);
        pout = out;
        for(i=0;i<lp_sampling;i++){
            pwindow = window_function[i];
            for(j=0;j<lp_sampling;j++){
                *(pout++) *= *pwindow;
            }
        }
        delete [] window_function;
    }
    return(1);
}

int get_best_candidate(
    int number_candidates,
    float *ftemp,
    int dim,
    int bits,
    float *in,
    int xdim,
    int ydim,
    int xdim_orig,
    int ydim_orig,
    int downsample,
    float *rotation,
    float *scale,
    float *x_trans,
    float *y_trans,
    float *template_real)
{
    int i,highest_i,j;
    float highest = -(float)1e20,xtrans,ytrans,value;

    for(i=0;i<number_candidates;i++){
        for(j=0;j<2;j++){
            /* rotate and scale suspect real image into ftemp */
            rotate_scale_translate_image(ftemp,dim,in_xdim,ydim,xdim_orig,ydim_orig,
                downsample,rotation[i]+(float)j*(float)180.0,scale[i]);
            realfft2d_in_place(ftemp,bits,0,wr,wi);
            gmft(template_real,ftemp,dim,bits,1,&xtrans,&ytrans,&value,1);
            if(value > highest){
                highest = value;
                highest_i = i;
                if(j==1)rotation[i] += (float)180.0;
                x_trans[i]=xtrans;
                y_trans[i]=ytrans;
            }
        }
    }
    rotation[0]=rotation[highest_i];
    scale[0]=scale[highest_i];
    x_trans[0]=x_trans[highest_i];
    y_trans[0]=y_trans[highest_i];
    return(1);
}

double log_id_remap(
    float *in,
    float *out,
    int dim)
{
    int i,dim2 = dim/2,xx;
    float *pin,*pout;
    double radius,fracx;
    double scale_increment_id;

    scale_increment_id=pow( 1.0/(double)START_RADIUS_ID, 1.0/(double)dim);
    pout = out;
    for(i=0;i<dim;i++){
        radius = (START_RADIUS_ID*(double)dim2) * pow(scale_increment_id,(double)i);
        xx = (int)radius;
        fracx = radius - (double)xx;
        pin = &in[xx];
    }
}

*pout = (float) ( (1.0-fracx) * (double)*(pin++) * (double)*(pin++) );
*(pout++) += (float) ( fracx*(double)*pin );
}

return(scale_increment_id);
}

int gmft_id(
    float *real1,
    float *imaginary1,
    float *real2,
    float *imaginary2,
    int dim,
    int bits,
    float *offset)
{
    int i,highest_i;
    float *preall1,*preall2,*pimaginary1,*pimaginary2;
    float mag1,mag2,dot,dott,cross,median[3],highest,ratio,ftmp;

    /* calculate phase differences and reload them into real1 and imaginary1 */
    /* keep phase differences to pi to -pi */
    preall1=real1,pimaginary1=imaginary1;
    preall2=real2,pimaginary2=imaginary2;
    for(i=0;i<dim;i++){
        mag1 = (float)sqrt( (double)( *preall1 * *preall1 + *pimaginary1 * *pimaginary1 ) );
        mag2 = (float)sqrt( (double)( *preall2 * *preall2 + *pimaginary2 * *pimaginary2 ) );
        if(mag1 == (float)0.0)mag1=(float)SMALL;
        if(mag2 == (float)0.0)mag2=(float)SMALL;
        dot = (*preall1 * *preall2 + *pimaginary1 * *pimaginary2)/mag1/mag2;
        dott = (float)1.0 - dot*dott;
        if(dott<(float)0.0)dott=(float)0.0;
        dott = (float)sqrt( (double)dott );
        cross = *preall1 * *pimaginary2 - *preall2 * *pimaginary1;
        if(cross < (float)0.0)cross = -(float)1.0;
        else cross = (float)1.0;
        ftmp = mag2;
        dott*ftmp/dott*=ftmp;
        *(preall1++) = dott;
        *(pimaginary1++) = cross*dott;
    }

    fft(real1,imaginary1,bits,1,wr,wi,1);

    /* search for highest value, then median find the center */
    highest = -(float)1e20;
    preall = real1;
    for(i=0;i<dim;i++){
        if( *preall > highest ){
            highest = *preall;
            highest_i = i;
        }
        preall++;
    }
    if(highest_i == 0){
        median[0]=real1[dim-1];
        median[1]=real1[0];
        median[2]=real1[1];
    }
    else if(highest_i == (dim-1)){
        median[0]=real1[dim-2];
        median[1]=real1[dim-1];
        median[2]=real1[0];
    }
    else {
        median[0]=real1[highest_i-1];
        median[1]=real1[highest_i];
        median[2]=real1[highest_i+1];
    }
    ratio = get_median_float(median);
    *offset = (float)highest_i * ratio;
    if( *offset > (float)dim/2.0 ) *offset -= (float)dim;
    return(1);
}

int refine_axis(
    unsigned char *ttemplate,
    int template_xdim,
    int template_ydim,
    unsigned char *suspect,
    int suspect_xdim,
    int suspect_ydim,
    float *x,
    float *y,
    int which)
{
}

```

```

unsigned char *psuspect;
int i,j,highest,fftdim,bits,xx,yy,xdim,ydim;
float x0,x1,x2,y0,y1,y2,*psuspect_integral,*ptemplate_integral;
float scan_x,scan_y,jump_x,jump_y,current_x,current_y;
float scale,translation,xdistance,ydistance,ydistance,suspect_dc,template_dc,frac;
double scale_increment_id;

/* first convert the y axis version to the x axis version */
x0 = x[0]; y0 = y[0];
if (which) {
    x1 = x[2]; y1 = y[2];
    x2 = x[1]; y2 = y[1];
    xdim = suspect_ydim;
    ydim = suspect_xdim;
} else {
    x1 = x[1]; y1 = y[1];
    x2 = x[2]; y2 = y[2];
    xdim = suspect_xdim;
    ydim = suspect_ydim;
}

/* determine the next highest power of two above higher of the two suspect axes */
if (suspect_xdim > suspect_ydim) highest = suspect_xdim;
else highest = suspect_ydim;
bits = 1 + (int) (log( (double) highest - 0.5 ) / log(2.0) );
fftdim = (int) pow(2.0, (double) bits + 0.00000001);

float *template_integral = new float[fftdim];
float *suspect_integral = new float[fftdim];
float *template_integral_imaginary = new float[fftdim];
float *suspect_integral_imaginary = new float[fftdim];
float *template_integral_copy = new float[fftdim];
float *suspect_integral_copy = new float[fftdim];

/* load suspect integral waveform */
psuspect_integral = suspect_integral;
for (j=0; j<fftdim; j++) { psuspect_integral[j] = (float) 0.0; }

/* load template integral */
psuspect = psuspect;
for (j=0; j<fftdim; j++) {
    psuspect_integral[j] = suspect_integral;
    for (i=0; i<xdim; i++) { psuspect_integral[i+j*xdim] = (float) 0.0; }
}

/* calculate scan elements that will be used in following stuff */
scan_x = (x1-x0)/(float) (xdim-1);
scan_y = (y1-y0)/(float) (ydim-1);
jump_x = (x2-x0)/(float) (xdim-1);
jump_y = (y2-y0)/(float) (ydim-1);

/* the next routines are split up since the one where the patch (suspect) is
outside the boundaries of the template forces boundary checking */
if (x[0]>0.0 && x[0]<=(float) (template_xdim-1) &&
    x[1]>0.0 && x[1]<=(float) (template_xdim-1) &&
    x[2]>0.0 && x[2]<=(float) (template_xdim-1) &&
    x[3]>0.0 && x[3]<=(float) (template_xdim-1) &&
    y[0]>0.0 && y[0]<=(float) (template_ydim-1) &&
    y[1]>0.0 && y[1]<=(float) (template_ydim-1) &&
    y[2]>0.0 && y[2]<=(float) (template_ydim-1) &&
    y[3]>0.0 && y[3]<=(float) (template_ydim-1) ) {
    ptemplate_integral = template_integral;
    for (j=0; j<fftdim; j++) { ptemplate_integral[j] = (float) 0.0; }
    current_x = x0 + (float) i * jump_x + (float) 0.5; // the addition of 0.5 is simply rounding
    current_y = y0 + (float) i * jump_y + (float) 0.5;
    ptemplate_integral = template_integral;
    for (j=0; j<xdim; j++) {
        yy = (int) current_y;
        xx = (int) current_x;
        *ptemplate_integral++ = (float) ttemplate[yy*template_xdim+xx];
        current_x += scan_x;
        current_y += scan_y;
    }
}

}
}
}

template_integral = template_integral;
for (j=0; j<fftdim; j++) { ptemplate_integral[j] = (float) 0.0; }
for (i=0; i<ydim; i++) {
    current_x = x0 + (float) i * jump_x + (float) 0.5; // the addition of 0.5 is simply rounding
    current_y = y0 + (float) i * jump_y + (float) 0.5;
    ptemplate_integral = template_integral;
    for (j=0; j<xdim; j++) {
        yy = (int) current_y;
        xx = (int) current_x;
        *ptemplate_integral++ = (float) ttemplate[yy*template_xdim+xx];
        current_x += scan_x;
        current_y += scan_y;
    }
}
}

}

template_dc = (float) 0.0;
ptemplate_integral = template_integral;
for (i=0; i<xdim; i++) { template_dc += *ptemplate_integral++; }
template_dc /= (float) xdim;
ptemplate_integral = template_integral;
for (i=0; i<xdim; i++) { ptemplate_integral[i] = template_dc; }
memcpy(template_integral_copy, template_integral, sizeof(float)*fftdim);

/* now perform a scale and translation matching of the two integrals */
window_id_vector(template_integral, xdim, fftdim);
window_id_vector(suspect_integral, xdim, fftdim);
memset(template_integral_imaginary, 0, sizeof(float)*fftdim);
memset(suspect_integral_imaginary, 0, sizeof(float)*fftdim);
fft(suspect_integral, suspect_integral_imaginary, bits, 0, wr, wi, 1);
fft(template_integral, template_integral_imaginary, bits, 0, wr, wi, 1);
// next routine places output into integral array
convert_to_magnitude_id_inplace(suspect_integral, suspect_integral_imaginary, fftdim);
// next routine places output into integral array
convert_to_magnitude_id_inplace(template_integral, template_integral_imaginary, fftdim);
// next routine places output into integral array
convert_to_magnitude_id_inplace(suspect_integral, suspect_integral_imaginary, fftdim);
// next routine places output into integral array
convert_to_magnitude_id_inplace(template_integral, template_integral_imaginary, fftdim);
// copy output back into fundamental array and zero out imaginary
memcpy(suspect_integral, suspect_integral_imaginary, sizeof(float)*fftdim);
memcpy(suspect_integral, suspect_integral_imaginary, sizeof(float)*fftdim);
memset(suspect_integral_imaginary, 0, sizeof(float)*fftdim);
memset(template_integral, template_integral_imaginary, sizeof(float)*fftdim);
memset(template_integral, template_integral_imaginary, sizeof(float)*fftdim);
// now do the id Fourier Mellin transform
window_id_vector(template_integral, fftdim, fftdim);
window_id_vector(suspect_integral, fftdim, fftdim);
fft(suspect_integral, suspect_integral_imaginary, bits, 0, wr, wi, 1);
fft(template_integral, template_integral_imaginary, bits, 0, wr, wi, 1);

/* gmfid to find any small scaling difference between the two */
gmfid(suspect_integral, suspect_integral_imaginary, template_integral,
    template_integral_imaginary, fftdim, bits, scale);
scale = (float) pow(scale_increment_id, (double) scale);

// update the x's and y's
xdistance = (x1-x0);
ydistance = (y1-y0);
x[3] += xdistance/(float) 1.0 - scale;
y[4] += ydistance/(float) 2.0; y[4] += ydistance/(float) 2.0;
if (which) {
    x[2] += xdistance; y[2] += ydistance;
    x1 = x[2]; y1 = y[2];
} else {
    x[1] += xdistance; y[1] += ydistance;
    x1 = x[1]; y1 = y[1];
}

/* now with the new scale information, perform a gmfid on the original and its rescaled counterpart */
ptemplate_integral = template_integral;
scale = (float) 1.0 / scale;
for (i=0, current_x=(float) 0.0; i<xdim; i++, current_x+=scale) {
    xx = (int) current_x;
    if (xx >= xdim-1) { ptemplate_integral++ = (float) 0.0; }
    else {
        frac = current_x - (float) xx;
        *ptemplate_integral = ((float) 1.0 - frac) * template_integral_copy[xx];
        *ptemplate_integral++ += frac * template_integral_copy[xx+1];
    }
}
}
}

```

```

// window the new scaled array; other one should be copy of windowed original
memcpy(suspect_integral,suspect_integral_copy,sizeof(float)*fftdim);
window_id_vector(template_integral_xdim,fftdim);
window_id_vector(suspect_integral_xdim,fftdim);
memset(suspect_integral_imaginary,0,sizeof(float)*fftdim);
memset(template_integral_imaginary,0,sizeof(float)*fftdim);
fft(suspect_integral,suspect_integral_imaginary,bits,0,wr,wi,1);
fft(template_integral,template_integral_imaginary,bits,0,wr,wi,1);

// now find the translation
gmf_id(suspect_integral,suspect_integral_imaginary,template_integral,
template_integral_imaginary,fftdim,bits,&translation);

// adjust x and y accordingly
translation *= (float)0.5; // I think this accounts for the fact that scaling has changed
origins????? very kludge
scan_x *= translation;
scan_y *= translation;
x[0] += scan_x; y[0] += scan_y;
x[1] += scan_x; y[1] += scan_y;
x[2] += scan_x; y[2] += scan_y;
x[3] += scan_x; y[3] += scan_y;
x[4] += scan_x; y[4] += scan_y;

delete [] template_integral;
delete [] suspect_integral;
delete [] template_integral_imaginary;
delete [] suspect_integral_imaginary;
delete [] template_integral_copy;
delete [] suspect_integral_copy;

return(0);
}

float refined_rotation(
float *x,
float *y,
unsigned char *suspect,
int suspect_xdim,
int suspect_ydim,
unsigned char *ttemplate,
int template_xdim,
int template_ydim)
{
int i,xx,yy,count,suspect;
float line_integral,template_count,suspect;
float line_integral_imaginary,template_count_imaginary,*pli,*pli_template;
float line_integral_imaginary_refined_rotation_dimension;
float line_integral_imaginary_refined_rotation_dimension;
float angle_x,suspect_y,suspect_x1,suspect_y1,suspect_dx,suspect_dy,suspect;
float template_y_refined_rotation_dimension;
float top_x,suspect_x(suspect_xdim-1),top_y,suspect_y(suspect_ydim-1);
float top_x_template(float) (template_xdim-1),top_y_template(float) (template_ydim-1);
float a_const,b_const,tweak,dc_suspect,dc_template;
float new_x,new_y,yaxis_x,axis_x,axis_y;
yaxis_x = (x[2]-x[0])/(float) (suspect_ydim-1); // this gives the unit vector in terms of the
yaxis_y = (y[2]-y[0])/(float) (suspect_ydim-1);
axis_x = (x[1]-x[0])/(float) (suspect_xdim-1);
axis_y = (y[1]-y[0])/(float) (suspect_xdim-1);

/* create line integral sweep around suspect's and template's center point */
pli = line_integral;
pli_template = line_integral_template;
dc_suspect = dc_template(float)0.0;
for(i=0;i<REFINED_ROTATION_DIMENSION;i++){
angle = (float)1 * (float)PI / (float)REFINED_ROTATION_DIMENSION;
x_suspect = x1_suspect = (float)0.5 + top_x_suspect/(float)2.0;
y_suspect = y1_suspect = (float)0.5 + top_y_suspect/(float)2.0;
dx_suspect = (float)sin((double)angle);
dy_suspect = (float)cos((double)angle);
x_suspect+=dx_suspect;x1_suspect-=dx_suspect;
y_suspect+=dy_suspect;y1_suspect-=dy_suspect;

x_template = x1_template = (float)0.5+x[4];
y_template = y1_template = (float)0.5+y[4];
dx_template = (axis_x*dx_suspect+axis_y*dy_suspect);
dy_template = (axis_y*dx_suspect+axis_x*dy_suspect);
x_template+=dx_template;x1_template-=dx_template;
y_template+=dy_template;y1_template-=dy_template;

*pli = (float)0.0;
*pli_template = (float)0.0;
count_template=0;count_suspect=0;
while(x_suspect>0.0 && x_suspect<top_x_suspect && y_suspect>0.0 && y_suspect<top_y_suspect){
xx = (int)x_suspect;
yy = (int)y_suspect;
*pli += suspect[yy*suspect_xdim+xx];
}
}

xx = (int)x1_suspect;
yy = (int)y1_suspect;
*pli += suspect[yy*suspect_xdim+xx];

x_suspect+=dx_suspect;x1_suspect-=dx_suspect;
y_suspect+=dy_suspect;y1_suspect-=dy_suspect;
count_suspect++;

if(y_template>0.0&&y_template<top_y_template&&x_template>0.0&&x_template<top_x_template)
&&y1_template>0.0&&y1_template<top_y_template&&x1_template>0.0&&x1_template<top_x_template){
xx = (int)x_template;
yy = (int)y_template;
*pli_template += ttemplate[yy*template_xdim+xx];

xx = (int)x1_template;
yy = (int)y1_template;
*pli_template += ttemplate[yy*template_xdim+xx];

x_template+=dx_template;x1_template-=dx_template;
y_template+=dy_template;y1_template-=dy_template;
count_template++;
}

*pli /= (float)count_suspect;
*pli_template /= (float)count_template;
dc_suspect += *pli++;
dc_template += *pli_template++;

/* now one-d fft them and one d gmf */
memset(line_integral_imaginary,0,sizeof(float)*REFINED_ROTATION_DIMENSION);
memset(line_integral_template_imaginary,0,sizeof(float)*REFINED_ROTATION_DIMENSION);
pli = line_integral;
pli_template = line_integral_template;
dc_suspect /= (float)REFINED_ROTATION_DIMENSION;
dc_template /= (float)REFINED_ROTATION_DIMENSION;
for(i=0;i<REFINED_ROTATION_DIMENSION;i++){
*pli++ -= dc_suspect;
*pli_template++ -= dc_template;
}

fft(line_integral,line_integral_imaginary,REFINED_ROTATION_BITS,0,wr,wi,1);
fft(line_integral_template,line_integral_template_imaginary,REFINED_ROTATION_BITS,0,wr,wi,1);

gmf_id(line_integral,line_integral_imaginary,line_integral_template,line_integral_template_imaginary,
REFINED_ROTATION_DIMENSION,REFINED_ROTATION_DIMENSION,REFINED_ROTATION_DIMENSION,
tweak * - ((float)180.0 / (float)REFINED_ROTATION_DIMENSION);

/* update xy0 thru xy3 */
a_const = (float)cos( (double)tweak * PI /180.0 );
b_const = (float)sin( (double)tweak * PI /180.0 );

new_x = a_const*(x[4]-x[0]) - b_const*(y[4]-y[0]);
new_y = b_const*(x[4]-x[0]) + a_const*(y[4]-y[0]);
x[0] = x[4] - new_x;
y[0] = y[4] - new_y;
new_x = a_const*(x[4]-x[1]) - b_const*(y[4]-y[1]);
new_y = b_const*(x[4]-x[1]) + a_const*(y[4]-y[1]);
x[1] = x[4] - new_x;
y[1] = y[4] - new_y;
new_x = a_const*(x[4]-x[2]) - b_const*(y[4]-y[2]);
new_y = b_const*(x[4]-x[2]) + a_const*(y[4]-y[2]);
x[2] = x[4] - new_x;
y[2] = y[4] - new_y;
new_x = a_const*(x[4]-x[3]) - b_const*(y[4]-y[3]);
new_y = b_const*(x[4]-x[3]) + a_const*(y[4]-y[3]);
x[3] = x[4] - new_x;
y[3] = y[4] - new_y;

return(tweak);
}

int Align::fine_tune_x_y(unsigned char *ttemplate,
int template_xdim,
int template_ydim,
unsigned char *suspect,
int suspect_xdim,
int suspect_ydim,
float *x,
float *y,
float *rotation)
{
//int foot=1;
float refinement;
}

```

```

axis_x = x[2]-x[0]/(float)(inydim-1); /* this gives the unit vector in terms of the suspect array */
axis_y = y[2]-y[0]/(float)(inydim-1);
axis_dist = (float)sqrt((double) (axis_x*axis_x+axis_y*axis_y));
axis_x = (x[1]-x[0])/((float)(inxdim-1));
axis_y = (y[1]-y[0])/((float)(inydim-1));
axis_dist = (float)sqrt((double) (axis_x*axis_x+axis_y*axis_y));

/* starts is origin dotted with axes */
x_start = (-x[0] * axis_x - y[0] * axis_y)/axis_dist/xaxis_dist;
y_start = (-x[0] * axis_x + y[0] * axis_y)/axis_dist/yaxis_dist;
scan_x = axis_x/axis_dist/xaxis_dist;
scan_y = axis_y/axis_dist/yaxis_dist;
jump_x = axis_x/axis_dist/xaxis_dist;
jump_y = axis_y/axis_dist/yaxis_dist;

pout = out;
for(i=0;i<outydin;i++){
    ii=(float)i;
    current_x = x_start + ii * jump_x;
    current_y = y_start + ii * jump_y;
    if(num_channels==1){
        for(j=0;j<outxdim;j++){
            if(current_x<(float)0.0 || current_y>(float)(inydim-1)){
                // option == 0)pout++; // this option preserves the rest of template
                else *(pout++) = (unsigned char)0;
            }
        }
    }
    else {
        xx = (int)current_x;
        yy = (int)current_y;
        frax = current_x - (float)xx ;
        fracx = current_x - (float)yy ;
        pin = sin(yy*inxdim + xx);
        ftmp = ((float)1.0-frax)*((float)1.0-fray)*(float)*pin;;
        ftmp += (fracx*((float)1.0-fray)*(float)*pin);
        pin += (inxdim-1);
        ftmp += (((float)1.0-frax)*fracy*(float)*pin++);
        ftmp += (fracx*fracy*(float)*pin);
        /* debug lines, use with option =0, then it draws a dashed line around suspect
        (inydim-2))* (pout++) =(unsigned char)0;
        */
        *(pout++) = (unsigned char)fTMP;
        current_x ++ scan_x;
        current_y ++ scan_y;
    }
}

else if(num_channels==3){
    for(j=0;j<outxdim;j++){
        if(current_x<(float)0.0 || current_y>(float)(inydim-1)||current_y<(float)0.0
           || (option == 0)pout+=3; // this option preserves the rest of template
        else *(pout++) = *(pout++) = *(pout++) = (unsigned char)0;
    }
}
else {
    xx = (int)current_x;
    yy = (int)current_y;
    frax = current_x - (float)xx ;
    fracx = current_x - (float)yy ;
    tmp1 = ((float)1.0 - frax) * ((float)1.0-fray );
    tmp2 = frax * ((float)1.0-fray);
    tmp3 = ((float)1.0 - frax) * fracy ;
    tmp4 = frax * fracy ;
    pin = sin(3*(yy*inxdim + xx));
    ftmp = ftmpl * (float)*pin;
    pin+=3;
    ftmp += (ftmp2 * (float)*pin);
    pin += 3*(inxdim-1);
    ftmp += (ftmp3 * (float)*pin ) ;
    pin+=3;
    ftmp += (ftmp4 * (float)*pin );
    *(pout++) = (unsigned char)fTMP;
    pin = sin(3*(yy*inxdim + xx)+1);
    ftmp = ftmpl * (float)*pin;
    pin+=3;
    ftmp += (ftmp2 * (float)*pin);
    pin += 3*(inxdim-1);
    ftmp += (ftmp3 * (float)*pin ) ;
    pin+=3;
    ftmp += (ftmp4 * (float)*pin );
    *(pout++) = (unsigned char)fTMP;
    pin = sin(3*(yy*inxdim + xx)+2);
    ftmp = ftmpl * (float)*pin;
    pin+=3;
    ftmp += (ftmp2 * (float)*pin);
    pin += 3*(inxdim-1);
    ftmp += (ftmp3 * (float)*pin ) ;
    pin+=3;
    ftmp += (ftmp4 * (float)*pin );
    *(pout++) = (unsigned char)fTMP;
}

if(option == 1){ // clear ttemplate array
    pout=out;
    for(i=0;i<(num_channels*outxdim*outydin); i++){*(pout++) = (unsigned char)0;}
}
```

```

/* assuming the inputs are both real only, then real 2D FFT each */
realfft2d_in_place(template_lp_real,lp_bits,0,wr,wl);
realfft2d_in_place(suspect_lp_real,lp_bits,0,wr,wl);

/* perform generalized matched filter on the two resulting arrays, outputting some number
of likely candidates, with their associated parameters */
gmf(template_lp_real,suspect_lp_real,lp_sampling,lp_bits,number_candidates,
rotation_scale,value,0);

// change units on rotation and scale for later stages
for(i=0;i<number_candidates;i++){
rotation[i] = ((float)180.0 / (float)lp_sampling); // converts to degrees
scale[i] = (float)pow((double)scale_increment,(double)scale[i]); // converts to
linear scale
}

/* now we have a series of candidates ( or 1, and we just need to get the rotation
and translation information ) wherein one of them should be
the correct one; this next routine sifts through all candidates, including both
the nominal rotation state and the state 180 degrees rotated from the nominal, and
finds which rotation, scale, and translation gives the highest matched filter
output; which then will be passed to the last fine tuning stage*/
// returns best candidate in first element of rotation, scale, x_trans, y_trans
get_best_candidate(number_candidates,ftemp,fftdim,lp_bits,suspect_copy,
1*(suspect_xdim-1)/downsample,1*(suspect_ydim-1)/downsample,suspect_xdim,
suspect_ydim,downsample,rotation,scale,x_trans,y_trans,template_real);

/* convert the scale/rotation/translation parameters of the downsampled arrays
into the x and y positions of the four corners of the suspect array, as projected
onto the template array. Precision in keeping track of the various coordinate systems
translates into final alignments to well better than a single pixel, especially
in light of the subtleties involved with downsampling. The four corners
are labelled 0 through 3 in the arrays x and y, where element 0 is the upper left corner
of the suspect, element 1 is the upper right, element 2 lower left, element 3 lower right.
The master 0 0 origin is placed at the upper left of the template array, while
the centerpoints of the two arrays play a role in rotations. The fifth, while
point in the x and y arrays is the centerpoint, used just so you don't have to
recalculate it all the time*/
get_corners_and_center(x,y,rotation[0],scale[0],x_trans[0],y_trans[0],
suspect_xdim,suspect_ydim,fftdim,downsample);

/* now fine tune the result using tricky tricks, see notebook of Nov 28, 1995 */
if(num_channels == 1){
fine_tune_x_y(template_xdim,template_ydim,suspect_xdim,
suspect_ydim,x,y,rotation);
}
else if(num_channels == 3){
fine_tune_x_y(template_lum,template_xdim,template_ydim,suspect_lum,suspect_xdim,
suspect_ydim,x,y,rotation);
}

/* last but not least, create the output image array, with various options */
final_image(template_xdim,template_ydim,template_ydim,suspect_xdim,
suspect_ydim,x,y,num_channels,1); // '1' stands for aligned suspect with black
everywhere else

/* Record some results of the alignment process in our status structure */
m_alignStatus.rotation = rotation[0];
m_alignStatus.x_scale = scale[0];
m_alignStatus.y_scale = scale[0];
m_alignStatus.x_trans = x_trans[0];
m_alignStatus.y_trans = y_trans[0];

/* free em all */
delete [] template_real;
delete [] template_lp_real;
delete [] suspect_real;
delete [] suspect_lp_real;
delete [] ftemp;
delete [] suspect_copy;
delete [] suspect_lum;
delete [] template_lum;

return(1);
}

/* shell to at least get the main registration program up and running, tested */
#ifdef NEED_MAIN
main()
//
// For Geoff's testing purposes, this main() function was used to
// create a stand-alone program which exercised the alignment
// algorithms. This is #ifdef'd out for the windows version.
//
main( int argc, char *argv[] )

```



```

{
    int template_xdim, template_ydim, suspect_xdim, suspect_ydim;
    char template_filename[80], suspect_filename[80];
    FILE *inf;

    printf("\nTemplate file name please: ");
    scanf("%s", template_filename);
    printf("\nx dimension and y dimension of template file: ");
    scanf("%d %d", &template_xdim, &template_ydim);
    printf("\nsuspect file name please: ");
    scanf("%s", suspect_filename);
    printf("\nx dimension and y dimension of suspect file: ");
    scanf("%d %d", &suspect_xdim, &suspect_ydim);

    unsigned char *img = new unsigned char[template_xdim*template_ydim*sizeof(unsigned char)];
    unsigned char *img1 = new unsigned char[suspect_xdim*suspect_ydim*sizeof(unsigned char)];

    /* read in binary data into template */
    inf = fopen(template_filename, "rb");
    if(!inf) {
        fprintf(stderr, "register: can't open %s\n", template_filename);
        exit(1);
    }
    fread(img, sizeof(unsigned char), template_xdim*template_ydim, inf);
    fclose(inf);

    inf = fopen(suspect_filename, "rb");
    if(!inf) {
        fprintf(stderr, "register: can't open %s\n", suspect_filename);
        exit(1);
    }
    fread(img1, sizeof(unsigned char), suspect_xdim*suspect_ydim, inf);
    fclose(inf);

    /* returns registered image inside array 'template' */
    direct_registration(img, template_xdim, template_ydim, img1, suspect_xdim, suspect_ydim);

    /* write out binary data from template */
    inf = fopen("reg_out", "wb");
    if(!inf) {
        fprintf(stderr, "register: can't open %s\n", "reg_out");
        exit(1);
    }
    fwrite(img, sizeof(unsigned char), template_xdim*template_ydim, inf);
    fclose(inf);

    /* free and clean up */
    delete [] img;
    delete [] img1;
    return(0);
}

#endif //NEED_MAIN

ALIGN_H
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// FILE: Align.h
//
// DESCRIPTION:
// Header file for the Alignment core algorithm code and the "Align"
// class used to encapsulate this code.
//
// The Alignment code is equivalent to Geoff Rhoads "Register" core
// algorithms, which were first created and run as a stand-alone C program
// on the SGI, then ported to Win95 and Visual C++ as a "console" program,
// and finally incorporated into the Signer windows application.
//
// Copyright (C) 1996 Digimarc Incorporated, all rights reserved.
//
// Define ALIGN_H
//
// A structure used to define results of the alignment process.
// typedef struct
// {
//     float rotation;
//     float x_scale;
//     float y_scale;
//     float x_trans;
//     float y_trans;
//     float refinement;
// } AlignStatus;
//
// Function prototypes: entry functions
// class Align
public:
    Align();
    int direct_registration(unsigned char *ttemplate,
        int template_xdim,
        int template_ydim,
        unsigned char *suspect,
        int suspect_xdim,
        int suspect_ydim,
        int num_channels);

    // Accessor for status
    const AlignStatus GetAlignStatus(void) const {return m_alignStatus;}

private:
    // Private structure which contains results of alignment
    AlignStatus m_alignStatus;

    int fine_tune_x_y(unsigned char *ttemplate,
        int template_xdim,
        int template_ydim,
        unsigned char *suspect,
        int suspect_xdim,
        int suspect_ydim,
        float *x,
        float *y,
        float *rotation);
};

// Function prototypes: private functions
int gmf_id(float *real1,
    float *imaginary1,
    float *real2,
    float *imaginary2,
    int dim,
    int bits,
    float *offset);

#endif // ALIGN_H

ALIGNDLG.CPP
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// AlignDlg.cpp : implementation file
//
#include "stdafx.h"
#include "signer.h"
#include "AlignDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// AlignDlg
//
IMPLEMENT_DYNAMIC(AlignDlg, CFileDialog)
AlignDlg::AlignDlg(BOOL bopenFileDialog, LPCTSTR lpszDefExt, LPCTSTR lpszFileName,
    DWORD dwFlags, LPCTSTR lpszFilter, CWnd* pParentWnd) :
    CFileDialog(bopenFileDialog, lpszDefExt, lpszFileName, dwFlags, lpszFilter,
        pParentWnd)
{
}

BEGIN_MESSAGE_MAP(AlignDlg, CFileDialog)
    //{{AFX_MSG_MAP(AlignDlg)
    // NOTE - The ClassWizard will add and remove mapping macros here.
    //}}AFX_MSG_MAP
    END_MESSAGE_MAP()
}

ALIGNDLG.H
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// AlignDlg.h : header file
//
// AlignDlg dialog
//
class AlignDlg : public CFileDialog
{
public:
    AlignDlg();
    int direct_registration(unsigned char *ttemplate,
        int template_xdim,
        int template_ydim,
        unsigned char *suspect,
        int suspect_xdim,
        int suspect_ydim,
        int num_channels);

    // Accessor for status
    const AlignStatus GetAlignStatus(void) const {return m_alignStatus;}

private:
    // Private structure which contains results of alignment
    AlignStatus m_alignStatus;

    int fine_tune_x_y(unsigned char *ttemplate,
        int template_xdim,
        int template_ydim,
        unsigned char *suspect,
        int suspect_xdim,
        int suspect_ydim,
        float *x,
        float *y,
        float *rotation);
};

// Function prototypes: private functions
int gmf_id(float *real1,
    float *imaginary1,
    float *real2,
    float *imaginary2,
    int dim,
    int bits,
    float *offset);

#endif // ALIGN_H

ALIGNDLG.CPP
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// AlignDlg.cpp : implementation file
//
#include "stdafx.h"
#include "signer.h"
#include "AlignDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// AlignDlg
//
IMPLEMENT_DYNAMIC(AlignDlg, CFileDialog)
AlignDlg::AlignDlg(BOOL bopenFileDialog, LPCTSTR lpszDefExt, LPCTSTR lpszFileName,
    DWORD dwFlags, LPCTSTR lpszFilter, CWnd* pParentWnd) :
    CFileDialog(bopenFileDialog, lpszDefExt, lpszFileName, dwFlags, lpszFilter,
        pParentWnd)
{
}

BEGIN_MESSAGE_MAP(AlignDlg, CFileDialog)
    //{{AFX_MSG_MAP(AlignDlg)
    // NOTE - The ClassWizard will add and remove mapping macros here.
    //}}AFX_MSG_MAP
    END_MESSAGE_MAP()
}

ALIGNDLG.H
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// AlignDlg.h : header file
//
// AlignDlg dialog
//
class AlignDlg : public CFileDialog
{
public:
    AlignDlg();
    int direct_registration(unsigned char *ttemplate,
        int template_xdim,
        int template_ydim,
        unsigned char *suspect,
        int suspect_xdim,
        int suspect_ydim,
        int num_channels);

    // Accessor for status
    const AlignStatus GetAlignStatus(void) const {return m_alignStatus;}

private:
    // Private structure which contains results of alignment
    AlignStatus m_alignStatus;

    int fine_tune_x_y(unsigned char *ttemplate,
        int template_xdim,
        int template_ydim,
        unsigned char *suspect,
        int suspect_xdim,
        int suspect_ydim,
        float *x,
        float *y,
        float *rotation);
};

// Function prototypes: private functions
int gmf_id(float *real1,
    float *imaginary1,
    float *real2,
    float *imaginary2,
    int dim,
    int bits,
    float *offset);

#endif // ALIGN_H

```

```

{
    DECLARE_DYNAMIC(AlignDlg)

public:
    AlignDlg(BOOL bOpenFileDialog, // TRUE for FileOpen, FALSE for FileSaveAs
        LPCTSTR lpszDefExt = NULL,
        LPCTSTR lpszFileName = NULL,
        DWORD dwFlags = OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
        LPCTSTR lpszFilter = NULL,
        CWnd* pParentWnd = NULL);

protected:
    //((AFX_MSG(AlignDlg)
    // NOTR - the ClassWizard will add and remove member functions here.
    //))AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////
// FILE: CoxKey.cpp
////////////////////////////////////
// DESCRIPTION:
// Contains the implementation of the CoExtensive Key class (CoxKey).
// A Coextensive key is also known as a "snowy image" or "code pattern".
// Copyright (C) 1996 Digimarc Corporation, all rights reserved.
// Include "CoxKey.h"
#include "CoxKey.h"
#include "dibapi.h"

////////////////////////////////////
// CoxKey()
////////////////////////////////////
// The constructor for the class takes a user key as the seed to the
// random number generator, a pointer to the BitmapInfo structure
// which defines the dimensions, etc. of the DIB, and a pointer to
// the DIB image space where we will put the snow. We basically
// seed the random number generator and fill the image data space
// with random values.
// Note that we must be careful to adhere to the core algorithms
// standard that the origin of an image is at the top left. Since
// Windows Bitmap images (DIBs) usually use the lower left as the
// origin, we need to be careful of the ordering and in the typical
// case fill the scan lines w/ random data from bottom to top.
CoxKey::CoxKey(unsigned user_key, BITMAPINFO *bmi, LPSTR lpDIBbits)
{
    char *p_line;
    int width_in_bytes, line_cnt, i, j, line;
    BOOL bottom_up;

    this->user_key = user_key; // save copy of the user's key
    image_data = lpDIBbits; // save huge ptr to image data

    // Set up a pointer to the BITMAPINFOHEADER and RGBQUAD array.
    bmiHeader = &bmi->bmiHeader;
    bmiColors = &bmi->bmiColors[0];

    // Set the pointer to the image data.
    this->lpDIBbits = lpDIBbits;

    // Check to see if this is in a format we handle (currently 8 bit only)
    // Need to throw and exception here.
    if (bmiHeader->biBitCount != 8 && bmiHeader->biBitCount != 24)
        return;

    width_in_bytes = (int) WIDTHBYTES(bmiHeader->biWidth * bmiHeader->biBitCount);

    // Seed the random number generator
    srand(user_key);

    // Image may be top to bottom or bottom to top.
    // We must generate snow accordingly
    if (bmiHeader->biHeight > 0)
    {
        bottom_up = TRUE;
        line = bmiHeader->biHeight - 1;
    }
    else
    {
        bottom_up = FALSE;
        line = 0;
    }
}

```

```

// Generate snow one image scan line at a time.
for (line_cnt = 0; line_cnt < bmiHeader->biHeight; line_cnt++)
{
    // Set pointer to first byte for this scan line.
    p_line = &image_data[line * (long) width_in_bytes];
    for (i = 0, j = 0; i < bmiHeader->biWidth; i++)
    {
        if (bmiHeader->biBitCount == 24)
        {
            // For 24 bit color case, need r,g,b snow...
            p_line[j++] = (char) rand();
            p_line[j++] = (char) rand();
            p_line[j++] = (char) rand();
        }
        else
        {
            // For test to make grey-scale and color keys match
            // we must call rand 3 times, but only keep same value
            // as the green channel of the rgb version. This way,
            // if we convert color image to greyscale we can read it.
            rand();
            p_line[i] = (char) rand(); // we make grey snow same as green.
            rand();
        }
    }
    if (bottom_up) line--;
    else line++;
}

void CoxKey::UseNewKey(unsigned newkey)
{
    char *p_line;
    int width_in_bytes, line_cnt, i;

    // Save the new key.
    user_key = newkey;

    width_in_bytes = (int) WIDTHBYTES(bmiHeader->biWidth * bmiHeader->biBitCount);

    // Seed the random number generator
    srand(user_key);

    for (line_cnt = 0; line_cnt < bmiHeader->biHeight; line_cnt++)
    {
        // Set pointer to first byte for this scan line.
        p_line = &image_data[line_cnt * (long) width_in_bytes];
        for (i = 0; i < bmiHeader->biWidth; i++)
        {
            p_line[i] = (char) rand();
        }
    }
}

////////////////////////////////////
// COXKEY.H
////////////////////////////////////
// FILE: CoxKey.h
//
// DESCRIPTION:
// The CoxKey (for Coextensive Key) class encapsulates the functions and
// data structures used to generate a "snowy image" of the same extent
// (i.e., x, y dimensions) as the input image.
//
// This header file should be included by any module which creates or
// makes use of coxkey objects.
//
// CREATION DATE: August 15, 1995
//
// Copyright (c) 1995 Digimarc Incorporated, all rights reserved.*
//*****
// #define COXKEY_H
// #define COXKEY_H
//
// #include "digimarc.h"
// #include "Params.h"
// #include "RawImage.h"
// #include "stdafx.h"
// #include "afx.h"
//
// class CoxKey
// {
//     // Public member functions
// public:
//     // The constructor is passed the user key value and ptrs to the DIB header

```

```

// structures and the data space. The header is assumed to be filled out
// correctly, while the data space is allocated but empty.
// Alternative: pass an HDIB handle, allowing this class to handle locking.
// FOR NOW, I ALSO ASSUME THE PALETTE HAS BEEN SET UP (its the same as image we are signing)
// CoXKey(int user_key, HDIB hDIB);
CoXKey(unsigned user_key, BITMAPINFO *bmi, LPSTR lpDIBbits);

```

```

// Private member functions
private:
// This function may be a useful idea for future, but it needs rework.
// I'm making it private to assure no one is calling it.
void UseNewKey(unsigned newkey);

```

```

// Private data
private:
// Copy of the user key value.
unsigned user_key;

// Pointers to the bitmap info header structure, and the palette array.
BITMAPINFOHEADER *bmiHeader; // Pts to beginning of palette array
RGBQUAD *pColorTable;

LPSTR lpDIBbits; // Pointer to DIB bits
char *image_data; // Pointer to raw image data.

```

```

};

#endif // COXKEY_H

```

DIPAPI.CPP

```

// dibapi.cpp
// Source file for Device-Independent Bitmap (DIB) API. Provides
// the following functions:
//
// PaintDIB() - Painting routine for a DIB
// CreateDIBPalette() - Creates a palette from a DIB
// FindDIBbits() - Returns a pointer to the DIB bits
// DIBwidth() - Gets the width of the DIB
// DIBheight() - Gets the height of the DIB
// PaletteSize() - Gets the size required to store the DIB's palette
// DIBnumColors() - Calculates the number of colors
// CopyHandle() - Makes a copy of the given global memory block
//
// This is a part of the Microsoft Foundation Classes C++ library.
// Copyright (C) 1992 Microsoft Corporation
// All rights reserved.
//
// This source code is only intended as a supplement to the
// Microsoft Foundation Classes Reference and Microsoft
// Quickhelp and/or Winhelp documentation provided with the library.
// See these sources for detailed information regarding the
// Microsoft Foundation Classes product.

```

```

#include "stdafx.h"
#include "dibapi.h"
#include <iostream>
#include <errno.h>

// PaintDIB()
// Parameters:
// * HDC hDC - DC to do output to
// * LPRECT lpDCRect - rectangle on DC to do output to
// * HDIB hDIB - handle to global memory with a DIB spec
// in it followed by the DIB bits
// * LPRECT lpDIBRect - rectangle of DIB to output into lpDCRect
// * CPalette* pPal - pointer to CPalette containing DIB's palette
// * Return Value:
// * BOOL - TRUE if DIB was drawn, FALSE otherwise
// * Description:
// * Painting routine for a DIB. Calls StretchDIBits() or

```

```

* SetDIBitsToDevice() to paint the DIB. The DIB is
* output to the specified DC, at the coordinates given
* in lpDCRect. The area of the DIB to be output is
* given by lpDIBRect.
*****
BOOL WINAPI PaintDIB(HDC hDC,
LPRECT lpDCRect,
HDIB hDIB,
LPRECT lpDIBRect,
CPalette* pPal)
{
LPSTR lpDIBhdr; // pointer to BITMAPINFOHEADER
LPSTR lpDIBbits; // pointer to DIB bits
BOOL bSuccess=FALSE;
HPALETTE hPal=NULL; // Our DIB's palette
HPALETTE holdPal=NULL; // Previous palette

// Check for valid DIB handle */
if (hDIB == NULL)
return FALSE;

// Lock down the DIB, and get a pointer to the beginning of the bit
// buffer
lpDIBhdr = (LPSTR)::GlobalLock((HGLOBAL) hDIB);
lpDIBbits = ::FindDIBbits(lpDIBhdr);

// Get the DIB's palette, then select it into DC
if (pPal != NULL)
{
hPal = (HPALETTE) pPal->m_hObject;

// Select as background since we have
// already realized in foreground if needed
holdPal = ::SelectPalette(hDC, hPal, TRUE);
}

// Make sure to use the stretching mode best for color pictures */
::SetStretchBltMode(hDC, COLORONCOLOR);

// Determine whether to call StretchDIBits() or SetDIBitsToDevice() */
if ((RECTWIDTH(lpDCRect) == RECTWIDTH(lpDIBRect)) &&
(RECTHEIGHT(lpDCRect) == RECTHEIGHT(lpDIBRect)))
bSuccess = ::SetDIBitsToDevice(hDC,
lpDCRect->left,
lpDCRect->top,
RECTWIDTH(lpDCRect),
RECTHEIGHT(lpDCRect),
lpDIBbits, 0,
(int)DIBheight(lpDIBhdr) -
lpDIBRect->top,
lpDIBRect->top,
RECTHEIGHT(lpDIBRect),
0,
(WORD)DIBheight(lpDIBhdr),
lpDIBbits,
(LPBITMAPINFO)lpDIBhdr,
DIB_RGB_COLORS);
else
bSuccess = ::StretchDIBits(hDC,
lpDCRect->left,
lpDCRect->top,
RECTWIDTH(lpDCRect),
RECTHEIGHT(lpDCRect),
lpDIBbits, 0,
(int)DIBheight(lpDIBhdr) -
lpDIBRect->top,
lpDIBRect->top,
RECTHEIGHT(lpDIBRect),
0,
(WORD)DIBheight(lpDIBhdr),
lpDIBbits,
(LPBITMAPINFO)lpDIBhdr,
DIB_RGB_COLORS,
SRCCOPY);

::GlobalUnlock((HGLOBAL) hDIB);

// Reselect old palette */
if (holdPal != NULL)
{
::SelectPalette(hDC, holdPal, TRUE);
}

return bSuccess;
}

/*****
* CreatedDIBPalette()
* Parameter:
*****

```



```

* height field if it is an other-style DIB.
*****
DWORD WINAPI DIBHeight(LPSTR lpDIB)
{
    LPBITMAPINFOHEADER lpbmi; // pointer to a Win 3.0-style DIB
    LPBITMAPCOREHEADER lpbmc; // pointer to an other-style DIB

    /* point to the header (whether old or Win 3.0 */
    lpbmi = (LPBITMAPINFOHEADER)lpDIB;
    lpbmc = (LPBITMAPCOREHEADER)lpDIB;

    /* return the DIB height if it is a Win 3.0 DIB */
    if (IS_WIN30_DIB(lpbmi))
        return lpbmi->biHeight;
    else /* it is an other-style DIB, so return its height */
        return (DWORD)lpbmc->bcHeight;
}

/* *****
* PaletteSize()
* Parameter:
* LPSTR lpbi - pointer to packed-DIB memory block
* Return Value:
* WORD - size of the color palette of the DIB
* Description:
* This function gets the size required to store the DIB's palette by
* multiplying the number of colors by the size of an RGBQUAD (for a
* Windows 3.0-style DIB) or by the size of an RGBTRIPLE (for an other-
* style DIB).
* ***** */

WORD WINAPI PaletteSize(LPSTR lpbi)
{
    /* calculate the size required by the palette */
    if (IS_WIN30_DIB(lpbi))
        return (WORD)((DIBNumColors(lpbi) * sizeof(RGBQUAD)));
    else
        return (WORD)((DIBNumColors(lpbi) * sizeof(RGBTRIPLE)));
}

/* *****
* DIBNumColors()
* Parameter:
* LPSTR lpbi - pointer to packed-DIB memory block
* Return Value:
* WORD - number of colors in the color table
* Description:
* This function calculates the number of colors in the DIB's color table
* by finding the bits per pixel for the DIB (whether Win3.0 or other-style
* DIB). If bits per pixel is 1, colors=2, if 4: colors=16, if 8: colors=256,
* if 24, no colors in color table.
* ***** */

WORD WINAPI DIBNumColors(LPSTR lpbi)
{
    WORD wBitCount; // DIB bit count

    /* If this is a Windows-style DIB, the number of colors in the
    * color table can be less than the number of bits per pixel
    * allows for (i.e. lpbi->biClrUsed can be set to some value).
    * If this is the case, return the appropriate value.
    */
    if (IS_WIN30_DIB(lpbi))
        return dwClrUsed;
}

```

```

if ((hCopy = (HANDLE) ::GlobalAlloc (GHND, dwLen)) != NULL)
{
    lpCopy = (BYTE *) ::GlobalLock((HGLOBAL) hCopy);
    lp = (BYTE *) ::GlobalLock((HGLOBAL) h);
    while (dwLen-- > 0)
        *lpCopy++ = *lp++;
    ::GlobalUnlock((HGLOBAL) hCopy);
    ::GlobalUnlock((HGLOBAL) h);
}

return hCopy;
}

// dibapi.h

// This is a part of the Microsoft Foundation Classes C++ library.
// Copyright (C) 1992 Microsoft Corporation
// All rights reserved.
//
// This source code is only intended as a supplement to the
// Microsoft Foundation Classes Reference and Microsoft
// QuickHelp and/or WinHelp documentation provided with the library.
// See these sources for detailed information regarding the
// Microsoft Foundation Classes product.

#ifdef INC_DIBAPI
#define INC_DIBAPI
/* Handle to a DIB */
DECLARE_HANDLE(HDIB);
/* DIB constants */
#define PALVERSION 0x300
/* DIB Macros */
#define IS_WIN32_DIB(lpbi) ((*LPWORD)(lpbi)) == sizeof(BITMAPINFOHEADER)
#define RECTWIDTH(lprc) ((lprc)->right - (lprc)->left)
#define RECTHEIGHT(lprc) ((lprc)->bottom - (lprc)->top)
/* WIDTHBYTES performs DWORD-aligning of DIB scanlines. The "bits"
parameter is the bit count for the scanline (bWidth * BitCount).
and this macro returns the number of DWORD-aligned bytes needed
to hold those bits.
#define WIDTHBYTES(bits) (((bits) + 31) / 32 * 4)
/* Function prototypes */
BOOL WINAPI PaintDIB (HDC, LPRECT, HDIB, LPRECT, CPalette* pPal);
BOOL WINAPI CreateDIBPalette(HDIB hDIB, CPalette* pPal);
LPSTR WINAPI FindDIBBits (LPSTR lpbi);
DWORD WINAPI DIBWidth (LPSTR lpbi);
WORD WINAPI DIBHeight (LPSTR lpbi);
WORD WINAPI DIBSize (LPSTR lpbi);
WORD WINAPI DIBNumColors (LPSTR lpbi);
HANDLE WINAPI DIBBitCount (LPSTR lpbi);
HANDLE WINAPI CopyHandle (HANDLE h);
BOOL WINAPI SaveDIB (HDIB hDIB, CFile& file);
HDIB WINAPI ReadDIBFile(CFile& file);
#endif // !INC_DIBAPI

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <memory.h>

#define MAX_LINEAR_DIMENSION 4096

/* generates ascii lines for t1-n */
main()
{
    int i;
    printf("\n\n");
    for(i=0; i<512; i++)
    {
        printf("%d", irvb(i,9));
    }
}

if (!i%16) printf("\n");
printf("\n\n");
for(i=0; i<1024; i++)
{
    printf("%d", irvb(i,10));
    if (!i%16) printf("\n");
}
}

static int t1[] = { 0, 1 };
static int t2[] = { 0, 2, 1, 3 };
static int t3[] = { 0, 4, 2, 6, 1, 5, 3, 7 };
static int t4[] = { 0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15 };
static int t5[] = { 0, 16, 8, 24, 4, 20, 12, 28, 2, 18, 10, 26, 6, 22, 14, 30, 1, 17, 9, 25, 5, 21, 13, 29, 3, 19, 11, 27, 7, 23, 15, 31 };
static int t6[] = { 0, 32, 16, 48, 8, 40, 24, 56, 4, 36, 20, 52, 12, 44, 28, 60, 2, 34, 18, 50, 10, 42, 26, 58, 6, 38, 22, 54, 14, 46, 30, 62, 1, 33, 17, 49, 9, 41, 25, 57, 5, 37, 21, 53, 13, 45, 29, 61, 3, 35, 19, 51, 11, 43, 27, 59, 7, 39, 23, 55, 15, 47, 31, 63 };
static int t7[] = { 0, 64, 32, 96, 16, 80, 48, 112, 8, 72, 40, 104, 24, 88, 56, 120, 4, 68, 36, 100, 20, 84, 52, 116, 12, 76, 44, 108, 28, 92, 60, 124, 2, 66, 34, 98, 18, 82, 50, 114, 10, 74, 42, 106, 26, 90, 58, 122, 6, 70, 38, 102, 22, 86, 54, 118, 14, 78, 46, 110, 30, 94, 62, 126, 1, 65, 33, 97, 17, 81, 49, 113, 9, 73, 41, 105, 25, 89, 57, 121, 5, 69, 37, 101, 21, 85, 53, 117, 13, 77, 45, 109, 29, 93, 61, 125, 3, 67, 35, 99, 19, 83, 51, 115, 11, 75, 43, 107, 27, 91, 59, 123, 7, 71, 39, 103, 23, 87, 55, 119, 15, 79, 47, 111, 31, 95, 63, 127 };
static int t8[] = { 0, 128, 64, 192, 32, 96, 224, 16, 144, 80, 208, 48, 176, 112, 240, 8, 136, 72, 200, 40, 168, 104, 232, 24, 152, 88, 216, 56, 184, 120, 248, 4, 132, 68, 196, 36, 164, 100, 228, 20, 148, 84, 212, 52, 180, 116, 244, 12, 140, 76, 204, 44, 172, 108, 236, 38, 156, 92, 220, 60, 188, 124, 252, 2, 130, 66, 194, 34, 162, 98, 226, 18, 146, 82, 210, 50, 178, 114, 242, 6, 138, 70, 202, 42, 170, 106, 234, 26, 154, 86, 218, 54, 182, 122, 246, 4, 134, 70, 206, 38, 166, 102, 230, 22, 150, 86, 214, 54, 182, 122, 246, 2, 132, 68, 198, 36, 164, 100, 228, 20, 148, 84, 212, 52, 180, 116, 244, 12, 140, 76, 204, 44, 172, 108, 236, 38, 156, 92, 220, 60, 188, 124, 252, 2, 130, 66, 194, 34, 162, 98, 226, 18, 146, 82, 210, 50, 178, 114, 242, 6, 138, 70, 202, 42, 170, 106, 234, 26, 154, 86, 218, 54, 182, 122, 246, 4, 134, 70, 206, 38, 166, 102, 230, 22, 150, 86, 214, 54, 182, 122, 246, 2, 132, 68, 198, 36, 164, 100, 228, 20, 148, 84, 212, 52, 180, 116, 244, 12, 140, 76, 204, 44, 172, 108, 236, 38, 156, 92, 220, 60, 188, 124, 252, 2, 130, 66, 194, 34, 162, 98, 226, 18, 146, 82, 210, 50, 178, 114, 242, 6, 138, 70, 202, 42, 170, 106, 234, 26, 154, 86, 218, 54, 182, 122, 246, 4, 134, 70, 206, 38, 166, 102, 230, 22, 150, 86, 214, 54, 182, 122, 246, 2, 132, 68, 198, 36, 164, 100, 228, 20, 148, 84, 212, 52, 180, 116, 244, 12, 140, 76, 204, 44, 172, 108, 236, 38, 156, 92, 220, 60, 188, 124, 252, 2, 130, 66, 194, 34, 162, 98, 226, 18, 146, 82, 210, 50, 178, 114, 242, 6, 138, 70, 202, 42, 170, 106, 234, 26, 154, 86, 218, 54, 182, 122, 246, 4, 134, 70, 206, 38, 166, 102, 230, 22, 150, 86, 214, 54, 182, 122, 246, 2, 132, 68, 198, 36, 164, 100, 228, 20, 148, 84, 212, 52, 180, 116, 244, 12, 140, 76, 204, 44, 172, 108, 236, 38, 156, 92, 220, 60, 188, 124, 252, 2, 130, 66, 194, 34, 162, 98, 226, 18, 146, 82, 210, 50, 178, 114, 242, 6, 138, 70, 202, 42, 170, 106, 234, 26, 154, 86, 218, 54, 182, 122, 246, 4, 134, 70, 206, 38, 166, 102, 230, 22, 150, 86, 214, 54, 182, 122, 246, 2, 132, 68, 198, 36, 164, 100, 228, 20, 148, 84, 212, 52, 180, 116, 244, 12, 140, 76, 204, 44, 172, 108, 236, 38, 156, 92, 220, 60, 188, 124, 252, 2, 130, 66, 194, 34, 162, 98, 226, 18, 146, 82, 210, 50, 178, 114, 242, 6, 138, 70, 202, 42, 170, 106, 234, 26, 154, 86, 218, 54, 182, 122, 246, 4, 134, 70, 206, 38, 166, 102, 230, 22, 150, 86, 214, 54, 182, 122, 246, 2, 132, 68, 198, 36, 164, 100, 228, 20, 148, 84, 212, 52, 180, 116, 244, 12, 140, 76, 204, 44, 172, 108, 236, 38, 156, 92, 220, 60, 188, 124, 252, 2, 130, 66, 194, 34, 162, 98, 226, 18, 146, 82, 210, 50, 178, 114, 242, 6, 138, 70, 202, 42, 170, 106, 234, 26, 154, 86, 218, 54, 182, 122, 246, 4, 134, 70, 206, 38, 166, 102, 230, 22, 150, 86, 214, 54, 182, 122, 246, 2, 132, 68, 198, 36, 164, 100, 228, 20, 148, 84, 212, 52, 180, 116, 244, 12, 140, 76, 204, 44, 172, 108, 236, 38, 156, 92, 220, 60, 188, 124, 252, 2, 130, 66, 194, 34, 162, 98, 226, 18, 146, 82, 210, 50, 178, 114, 242, 6, 138, 70, 202, 42, 170, 106, 234, 26, 154, 86, 218, 54, 182, 122, 246, 4, 134, 70, 206, 38, 166, 102, 230, 22, 150, 86, 214, 54, 182, 122, 246, 2, 132, 68, 198, 36, 164, 100, 228, 20, 148, 84, 212, 52, 180, 116, 244, 12, 140, 76, 204, 44, 172, 108, 236, 38, 156, 92, 220, 60, 188, 124, 252, 2, 130, 66, 194, 34, 162, 98, 226, 18, 146, 82, 210, 50, 178, 114, 242, 6, 138, 70, 202, 42, 170, 106, 234, 26, 154, 86, 218, 54, 182, 122, 246, 4, 134, 70, 206, 38, 166, 102, 230, 22, 150, 86, 214, 54, 182, 122, 246, 2, 132, 68, 198, 36, 164, 100, 228, 20, 148, 84, 212, 52, 180, 116, 244, 12, 140, 76, 204, 44, 172, 108, 236, 38, 156, 92, 220, 60, 188, 124, 252, 2, 130, 66, 194, 34, 162, 98, 226, 18, 146, 82, 210, 50, 178, 114, 242, 6, 138, 70, 202, 42, 170, 106, 234, 26, 154, 86, 218, 54, 182, 122, 246, 4, 134, 70, 206, 38, 166, 102, 230, 22, 150, 86, 214, 54, 182, 122, 246, 2, 132, 68, 198, 36, 164, 100, 228, 20, 148, 84, 212, 52, 180, 116, 244, 12, 140, 76, 204, 44, 172, 108, 236, 38, 156, 92, 220, 60, 188, 124, 252, 2, 130, 66, 194, 34, 162, 98, 226, 18, 146, 82, 210, 50, 178, 114, 242, 6, 138, 70, 202, 42, 170, 106, 234, 26, 154, 86, 218, 54, 182, 122, 246, 4, 134, 70, 206, 38, 166, 102, 230, 22, 150, 86, 214, 54, 182, 122, 246, 2, 132, 68, 198, 36, 164, 100, 228, 20, 148, 84, 212, 52, 180, 116, 244, 12, 140, 76, 204, 44, 172, 108, 236, 38, 156, 92, 220, 60, 188, 124, 252, 2, 130, 66, 194, 34, 162, 98, 226, 18, 146, 82, 210, 50, 178, 114, 242, 6, 138, 70, 202, 42, 170, 106, 234, 26, 154, 86, 218, 54, 182, 122, 246, 4, 134, 70, 206, 38, 166, 102, 230, 22, 150, 86, 214, 54, 182, 122, 246, 2, 132, 68, 198, 36, 164, 100, 228, 20, 148, 84, 212, 52, 180, 116, 244, 12, 140, 76, 204, 44, 172, 108, 236, 38, 156, 92, 220, 60, 188, 124, 252, 2, 130, 66, 194, 34, 162, 98, 226, 18, 146, 82, 210, 50, 178, 114, 242, 6, 138, 70, 202, 42, 170, 106, 234, 26, 154, 86, 218, 54, 182, 122, 246, 4, 134, 70, 206, 38, 166, 102, 230, 22, 150, 86, 214, 54, 182, 122, 246, 2, 132, 68, 198, 36, 164, 100, 228, 20, 148, 84, 212, 52, 180, 116, 244, 12, 140, 76, 204, 44, 172, 108, 236, 38, 156, 92, 220, 60, 188, 124, 252, 2, 130, 66, 194, 34, 162, 98, 226, 18, 146, 82, 210, 50, 178, 114, 242, 6, 138, 70, 202, 42, 170, 106, 234, 26, 154, 86, 218, 54, 182, 122, 246, 4, 134, 70, 206, 38, 166, 102, 230, 22, 150, 86, 214, 54, 182, 122, 246, 2, 132, 68, 198, 36, 164, 100, 228, 20, 148, 84, 212, 52, 180, 116, 244, 12, 140, 76, 204, 44, 172, 108, 236, 38, 156, 92, 220, 60, 188, 124, 252, 2, 130, 66, 194, 34, 162, 98, 226, 18, 146, 82, 210, 50, 178, 114, 242, 6, 138, 70, 202, 42, 170, 106, 234, 26, 154, 86, 218, 54, 182, 122, 246, 4, 134, 70, 206, 38, 166, 102, 230, 22, 150, 86, 214, 54, 182, 122, 246, 2, 132, 68, 198, 36, 164, 100, 228, 20, 148, 84, 212, 52, 180, 116, 244, 12, 140, 76, 204, 44, 172, 108, 236, 38, 156, 92, 220, 60, 188, 124, 252, 2, 130, 66, 194, 34, 162, 98, 226, 18, 146, 82, 210, 50, 178, 114, 242, 6, 138, 70, 202, 42, 170, 106, 234, 26, 154, 86, 218, 54, 182, 122, 246, 4, 134, 70, 206, 38, 166, 102, 230, 22, 150, 86, 214, 54, 182, 122, 246, 2, 132, 68, 198, 36, 164, 100, 228, 20, 148, 84, 212, 52, 180, 116, 244, 12, 140, 76, 204, 44, 172, 108, 236, 38, 156, 92, 220, 60, 188, 124, 252, 2, 130, 66, 194, 34, 162, 98, 226, 18, 146, 82, 210, 50, 178, 114, 242, 6, 138, 70, 202, 42, 170, 106, 234, 26, 154, 86, 218, 54, 182, 122, 246, 4, 134, 70, 206, 38, 166, 102, 230, 22, 150, 86, 214, 54, 182, 122, 246, 2, 132, 68, 198, 36, 164, 100, 228, 20, 148, 84, 212, 52, 180, 116, 244, 12, 140, 76, 204, 44, 172, 108, 236, 38, 156, 92, 220, 60, 188, 124, 252, 2, 130, 66, 194, 34, 162, 98, 226, 18, 146, 82, 210, 50, 178, 114, 242, 6, 138, 70, 202, 42, 170, 106, 234, 26, 154, 86, 218, 54, 182, 122, 246, 4, 134, 70, 206, 38, 166, 102, 230, 22, 150, 86, 214, 54, 182, 122, 246, 2, 132, 68, 198, 36, 164, 100, 228, 20, 148, 84, 212, 52, 180, 116, 244, 12, 140, 76, 204, 44, 172, 108, 236, 38, 156, 92, 220, 60, 188, 124, 252, 2, 130, 66, 194, 34, 162, 98, 226, 18, 146, 82, 210, 50, 178, 114, 242, 6, 138, 70, 202, 42, 170, 106, 234, 26, 154, 86, 218, 54, 182, 122, 246, 4, 134, 70, 206, 38, 166, 102, 230, 22, 150, 86, 214, 54, 182, 122, 246, 2, 132, 68, 198, 36, 164, 100, 228, 20, 148, 84, 212, 52, 180, 116, 244, 12, 140, 76, 204, 44, 172, 108, 236, 38, 156, 92, 220, 60, 188, 124, 252, 2, 130, 66, 194, 34, 162, 98, 226, 18, 146, 82, 210, 50, 178, 114, 242, 6, 138, 70, 202, 42, 170, 106, 234, 26, 154, 86, 218, 54, 182, 122, 246, 4, 134, 70, 206, 38, 166, 102, 230, 22, 150, 86, 214, 54, 182, 122, 246, 2, 132, 68, 198, 36, 164, 100, 228, 20, 148, 84, 212, 52, 180, 116, 244, 12, 140, 76, 204, 44, 172, 108, 236, 38, 156, 92, 220, 60, 188, 124, 252, 2, 130, 66, 194, 34, 162, 98, 226, 18, 146, 82, 210, 50, 178, 114, 242, 6, 138, 70, 202, 42, 170, 106, 234, 26, 154, 86, 218, 54, 182, 122, 246, 4, 134, 70, 206, 38, 166, 102, 230, 22, 150, 86, 214, 54, 182, 122, 246, 2, 132, 68, 198, 36, 164, 100, 228, 20, 148, 84, 212, 52, 180, 116, 244, 12, 140, 76, 204, 44, 172, 108, 236, 38, 156, 92, 220, 60, 188, 124, 252, 2, 130, 66, 194, 34, 162, 98, 226, 18, 146, 82, 210, 50, 178, 114, 242, 6, 138, 70, 202, 42, 170, 106, 234, 26, 154, 86, 218, 54, 182, 122, 246, 4, 134, 70, 206, 38, 166, 102, 230, 22, 150, 86, 214, 54, 182, 122, 246, 2, 132, 68, 198, 36, 164, 100, 228, 20, 148, 84, 212, 52, 180, 116, 244, 12, 140, 76, 204, 44, 172, 108, 236, 38, 156, 92, 220, 60, 188, 124, 252, 2, 130, 66, 194, 34, 162, 98, 226, 18, 146, 82, 210, 50, 178, 114, 242, 6, 138, 70, 202, 42, 170, 106, 234, 26, 154, 86, 218, 54, 182, 122, 246, 4, 134, 70, 206, 38, 166, 102, 230, 22, 150, 86, 214, 54, 182, 122, 246, 2, 132, 68, 198, 36, 164, 100, 228, 20, 148, 84, 212, 52, 180, 116, 244, 12, 140, 76, 204, 44, 172, 108, 236, 38, 156, 92, 220, 60, 188, 124, 252, 2, 130, 66, 194, 34, 162, 98, 226, 18, 146, 82, 210, 50, 178, 114, 242, 6, 138, 70, 202, 42, 170, 106, 234, 26, 154, 86, 218, 54, 182, 122, 246, 4, 134, 70, 206, 38, 166, 102, 230, 22, 150, 86, 214, 54, 182, 122, 246, 2, 132, 68, 198, 36, 164, 100, 228, 20, 148, 84, 212, 52, 180, 116, 244, 12, 140, 76, 204, 44, 172, 108, 236, 38, 156, 92, 220, 60, 188, 124, 252, 2, 130, 66, 194, 34, 162, 98, 226, 18, 146, 82, 210, 50, 178, 114, 242, 6, 138, 70, 202, 42, 170, 106, 234, 26, 154, 86, 218, 54, 182, 122, 246, 4, 134, 70, 206, 38, 166, 102, 230, 22, 150, 86, 214, 54, 182, 122, 246, 2, 132, 68, 198, 36, 164, 100, 228, 20, 148, 84, 212, 52, 180, 116, 244, 12, 140, 76, 204, 44, 172, 108, 236, 38, 156, 92, 220, 60, 188, 124, 252, 2, 130, 66, 194, 34, 162, 98, 226, 18, 146, 82, 210, 50, 178, 114, 242, 6, 138, 70, 202, 42, 170, 106, 234, 26, 154, 86, 218, 54, 182, 122, 246, 4, 134, 70, 206, 38, 166, 102, 230, 22, 150, 86, 214, 54, 182, 122, 246, 2, 132, 68, 198, 36, 164, 100, 228, 20, 148, 84, 212, 52, 180, 116, 244, 12, 140, 76, 204, 44, 172, 108, 236, 38, 156, 92, 220, 60, 188, 124, 252, 2, 130, 66, 194, 34, 162, 98, 226, 18, 146, 82, 210, 50, 178, 114, 242, 6, 138, 70, 202, 42, 170, 106, 234, 26, 154, 86, 218, 54, 182, 122, 246, 4, 134, 70, 206, 38, 166, 102, 230, 22, 150, 86, 214, 54, 182, 122, 246, 2, 132, 68, 198, 36, 164, 100, 228, 20, 148, 84, 212, 52, 180, 116, 244, 12, 140, 76, 204, 44, 172, 108, 236, 38, 156, 92, 220, 60, 188, 124, 252, 2, 130, 66, 194, 34, 162, 98, 226, 18, 146, 82, 210, 50, 178, 114, 242, 6, 138, 70, 202, 42, 170, 106, 234, 26, 154, 86, 218, 54, 182, 122, 246, 4, 134, 70, 206, 38, 166, 102, 230, 22, 150, 86, 214, 54, 182, 122, 246, 2, 132, 68, 198, 36, 164, 100, 228, 20, 148, 84, 212, 52, 180, 116, 244, 12, 140, 76, 204, 44, 172, 108, 236, 38, 156, 92, 220, 60, 188, 124, 252, 2, 130, 66, 194, 34, 162, 98,
```

```

case 6 : return( t6[nm] ) ;
case 7 : return( t7[nm] ) ;
case 8 : return( t8[nm] ) ;
case 9 : return( t9[nm] ) ;
case 10 : return( t10[nm] ) ;
default:
{
    r = 0 ;
    for( i = 0 ; i < nb ; i++ )
    {
        r = r << 1 ;
        r = r | ( nm & 1 ) ;
        nm = nm >> 1 ;
    }
    return( r ) ;
}

/* ffft() is a routine that calculates the discrete Fourier transform
of two arrays taken to be the real and the imaginary parts of an
complex array. It returns the transform in the arrays. */
void fff(float *ar,float *ai,int nbits,int inv,float *wr,float *wi,int neww)
/* float *ar ; /* the real part of the array */
/* float *ai ; /* the imag part of the array */
/* int nbits ; /* log base 2 of the number of elements in the arrays */
/* int inv ; /* nonzero to indicate the inverse transform */
/* float *wr ; /* the real part of an array of coefficients */
/* float *wi ; /* the imag part of an array of coefficients */
/* int neww ; /* nonzero to indicate the coefficients must be calced */
{
    register float *aar ;
    register float *aa ;
    register float *pi1 ;
    register float *pi2 ;
    register float *pi3 ;
    register float *r1 ;
    register float *r2 ;
    register float *i1 ;
    register float *i2 ;
    register int j ;
    register int n ;
    float fn ;
    float tpin ;
    register int n2 ;
    register int n1 ;
    register int nb ;
    int nblock ;
    register int nsep ;
    register int nsep2 ;
    register int ns ;
    register float areal ;
    register float aimag ;
    register float wreal ;
    register float winag ;
    register float *pwr ;
    register float *phi ;
    float w ;
    aar = ar ;
    aa = ai ;
    n = 1 << nbits ;
    fn = (float) n ;
    if( inv == 0 )
    {
        for( i = 0 ; i < n ; i++ )
        {
            aar[i] = aar[i] / fn ;
            aa[i] = -aa[i] / fn ;
        }
    }
    if( neww != 0 )
    {
        tpin = (float) 6.283186 / fn ;
        n2 = n / 2 ;
        for( nb = 0 ; nb < n2 ; nb++ )
        {
            w = tpin * ( (float) irvb( nb, nbits-1 ) ) ;
            wr[nb] = (float) cos( (double) w ) ;
            wi[nb] = (float) sin( (double) w ) ;
        }
    }
}

```

```

    }
    nblock = 1;
    nsep = n;
    for( ns = 0; ns < nbits; ns++)
    {
        nsep2 = nsep;
        nsep = nsep / 2;
        pwr = wr;
        pwi = wi;
        for( nb=0; nb < nblock; nb++, pwr++, pwi++)
        {
            n1 = nb*nsep2;
            n2 = n1*nsep;
            pr1 = &ar[n1];
            pr2 = &ar[n2];
            p11 = &aai[n1];
            p12 = &aai[n2];
            wreal = *pwr;
            wimag = *pwi;
            for( j=0; j<nsep; j++)
            {
                r1 = *pr1; r2 = *pr2; i1 = *p11; i2 = *p12;
                areal = wreal * r1 - wimag * i2;
                aimag = wimag * r1 + wreal * i2;
                *(pr2++) = r1 - areal;
                *(pi2++) = i1 - aimag;
                *(pr1++) = r1 + areal;
                *(pi1++) = i1 + aimag;
            }
        }
        nblock = nblock*2;
    }
    for( i = 0; i < n; i++)
    {
        j = irvb( i, nbits );
        if( i < j )
        {
            areal = aar[i];
            aimag = aai[i];
            areal = aai[j];
            aai[i] = aar[j];
            aai[i] = aai[j];
            aar[j] = areal;
            aai[j] = aimag;
        }
        if( inv == 0 ) aai[i] = -aai[i];
    }
    int fft2d(float *ar,float *ai,int nbits,int inv,float *wr,float *wi )
    {
        int i;
        int j;
        int ij;
        int ji;
        int n;
        float xr;
        float xi;
        n = 1 << nbits;
        for( i = 1; i < n; i++)
        {
            for( j = 0; j < i; j++)
            {
                ij = (i<<nbits)+j;
                ji = (j<<nbits)+i;
                xr = ar[ij];
                xi = ai[ij];
                ar[ji] = ar[ij];
                ai[ji] = ai[ij];
                ar[ij] = xr;
                ai[ij] = xi;
            }
        }
        fft( &ar[0], &ai[0], nbits, inv, wr, wi, 1 );
        for( i = 1; i < n; i++)
        {
            fft( &ar[i<<nbits], &ai[i<<nbits], nbits, inv, wr, wi, 0 );
        }
    }
    for( i = 1; i < n; i++)
    {
        for( j = 0; j < i; j++)
        {
            ij = (i<<nbits)+j;
            ji = (j<<nbits)+i;
            xr = ar[ij];
            xi = ai[ij];
            ar[ji] = ar[ij];
            ai[ji] = ai[ij];
            ar[ij] = xr;
            ai[ij] = xi;
        }
    }
    void realfft_two_arrays(float *array1,float *array2,int nbits,int inv,float *wr,float *wi,int
neww)
    {
        register int j;
        register int nhalf;
        float tempi[MAX_LINEAR_DIMENSION],temp2[MAX_LINEAR_DIMENSION];
        register float *ptempi;
        register float *ptemp2;
        register float *par;
        register float *pai;
        register float *pai1;
        register float *pai2;
        register float *ptemp1_1;
        register float *ptemp2_1;
        n = 1 << nbits;
        nhalf = n/2;
        if(!inv)
        {
            fft(array1,array2,nbits,inv,wr,wi,neww);
            /* sort the results */
            ptemp1 = tempi;
            ptemp2 = temp2;
            par = array1;
            pai = array2;
            *ptemp1 = *(par++);
            *ptemp2 = *(pai++);
            par1 = &array1[n-1];
            pai1 = &array2[n-1];
            ptemp1+=2;
            ptemp2+=2;
            for( j=1; j<nhalf; j++)
            {
                *(ptemp1++) = (float)0.5 * (*par + *par1);
                *(ptemp2++) = (float)0.5 * (*pai + *pai1);
                *(ptemp1++) = (float)0.5 * (*par - *pai1);
                *(ptemp2++) = (float)0.5 * (-*par + *par1);
                par++,par1--;pai++,pai1--;
            }
            tempi[1] = *par;
            temp2[1] = *pai;
            /* now copy the results back into original arrays */
            memcpy(array1,tempi,n*sizeof(float));
            memcpy(array2,temp2,n*sizeof(float));
        }
        else /* re-sort results */
        {
            ptemp1 = tempi;
            ptemp2 = temp2;
            par = array1;
            pai = array2;
            *ptemp1++ = *par;
            *ptemp2++ = *pai;
            par+=2;
            pai+=2;
            ptemp1_1 = &tempi[n-1];
            ptemp2_1 = &temp2[n-1];
            for( j=1; j<(n/2); j++)
            {
                *(ptemp1++) = (*par - *pai1);
                *(ptemp2++) = (*par + *pai1);
                *(ptemp1++) = (*par1 + *pai);
                *(ptemp2++) = (-*par1 + *pai);
                par+=2;
                pai+=2;
            }
            ptemp1 = array1[1];
            ptemp2 = array2[1];
        }
    }

```



```

fft(array1,array2,nbits,inv,wr,wi,neww);
}

/* this routine requires that the input array have two more rows of n appended, into which the nyquist
row will be placed */
int realfft2d_in_place(float *ar,int nbits,int inv,float *wr,float *wi )
{
    register int i ;
    register int j ;
    register int i1 ;
    register int j1 ;
    register int n ;
    register int n1 ;
    register int n2 ;
    register int nhalf ;
    register float xr ;
    register float xi ;
    register float x1 ;
    register float x11 ;
    float temp_r[MAX_LINEAR_DIMENSION],temp_i[MAX_LINEAR_DIMENSION];
    register float *ptemp_r;
    register float *ptemp_i;
    register float *par;
    register float *pai;
    register float *parl;
    register float *pail;
    register float *ptemp_r1;
    register float *ptemp_i1;

    n = 1 << nbits ;
    n2 = n*2;
    nhalf = n/2;

    if( !inv )
    {
        /* pre-transpose */
        for( i = 1 ; i < n ; i++ )
        {
            for( j = 0 ; j < i ; j++ )
            {
                i1 = (i<nbits)+j ;
                j1 = (j<nbits)+i ;
                xr = ar[i1] ;
                ar[i1] = ar[j1] ;
                ar[j1] = xr ;
            }
        }

        for( i = 0 ; i < nhalf ; i++ )
        {
            if(i==0)fft( ar[0], ar[n], nbits, inv, wr, wi, 1 ) ;
            else fft( ar[n2+i], ar[n2+i*n], nbits, inv, wr, wi, 0 ) ;

            /* sort and pack results */
            ptemp_r = temp_r;
            ptemp_i = temp_i[2];
            par = ar[n2+i*n];
            parl = ar[n2+i*n];
            *(ptemp_r++) = *(par++);
            *(ptemp_r++) = *(parl--);

            pai = ar[i+n2+i*n];
            pail = ar[n2+i+n2-1];
            for(j=1;j<nhalf;j++){
                *(ptemp_r++) = (float)0.5 * (*par + *parl);
                *(ptemp_r++) = (float)0.5 * (*pai + *pail);
                *(ptemp_i++) = (float)0.5 * (*pai - *pail);
                *(ptemp_i++) = (float)0.5 * (-*par + *parl);
                par++;parl--;pai++;pail--;
            }
            temp_i[0] = *par;
            temp_i[1] = *pai;

            /* now copy the results back into original arrays */
            memcpy( ar[n2+i],temp_r,n*sizeof(float));
            memcpy( ar[n2+i*n],temp_i,n*sizeof(float));
        }

        /* transpose */
        for( i = 2 ; i < n ; i+=2 ) {
            for( j = 0 ; j < i ; j+=2 ) {
                i1 = (i<nbits)+j ;
                j1 = (j<nbits)+i ;
                xr = ar[i1] ;
                xi = ar[j1] ;
                x1 = ar[i1+n] ;
                x11 = ar[j1+n] ;
                ar[i1] = ar[i1+n] ;
                ar[j1] = ar[j1+n] ;
                ar[i1+n] = xi ;
                ar[j1+n] = x1 ;
            }
        }

        /* undo format */
        for(i=0;i<(n/2);i++){
            memcpy(temp_r,ar[i*n],(n/2)*sizeof(float));
            memcpy(temp_i,ar[n/2+i*n],(n/2)*sizeof(float));
            memcpy(ar[i*n],temp_r,(n/2)*sizeof(float));
            memcpy(ar[n/2+i*n],temp_i,(n/2)*sizeof(float));
        }

        fft( ar[0], ar[n], nbits, inv, wr, wi, 1 ) ;
        for( i = 1 ; i < (1+n/2) ; i++ ) fft( ar[(2*i-1)*n], ar[(2*i+1)*n], nbits, inv, wr, wi, 0 ) ;
    }

    memcpy( ar[n], ar[n*n],n*sizeof(float));

    /* transpose */
    for( i = 2 ; i < n ; i+=2 ) {
        for( j = 0 ; j < i ; j+=2 ) {
            i1 = (i<nbits)+j ;
            j1 = (j<nbits)+i ;
            xr = ar[i1] ;
            xi = ar[j1] ;
            x1 = ar[i1+n] ;
            x11 = ar[j1+n] ;
            ar[i1] = ar[j1+n] ;
            ar[j1] = ar[i1+n] ;
            ar[i1+n] = xi ;
            ar[j1+n] = x1 ;
        }
    }

    /* re-sort results */
    for( i = 0 ; i < (n/2) ; i++ )
    {
        ptemp_r = temp_r;
        ptemp_i = temp_i;
        par = ar[(2*i)*n];
        *(ptemp_r++) = *(par++);
        *(ptemp_i++) = *(par++);

        pai = ar[2*(2*i+1)*n];
        ptemp_r1 = &temp_r[n-1];
        ptemp_i1 = &temp_i[n-1];
        for(j=1;j<(n/2);j++){
            *(ptemp_r++) = (*par - *pai+1) ;
            *(ptemp_r1--) = (*par + *pai+1) ;
            *(ptemp_i++) = (*par+1) + *pai ;
            *(ptemp_i1--) = (*par+1) - *pai ;
            par+=2;
            pai+=2;
        }
        *ptemp_r = ar[(2*i+1)*n];
        *ptemp_i = ar[(2*i+1)*n + 1];

        /* now copy the results back into original arrays */
        memcpy( ar[(2*i)*n],temp_r,n*sizeof(float));
        memcpy( ar[(1+2*i)*n],temp_i,n*sizeof(float));
    }

    fft( ar[(2*i)*n], ar[(2*i+1)*n], nbits, inv, wr, wi, 0 ) ;

    /* post transpose */

```

```

    m_BitsPerPixel = m_lpBmiHeader->biBitCount;
    m_XDim = m_lpBmiHeader->biWidth;
    m_YDim = m_lpBmiHeader->biHeight;
    m_Compression = m_lpBmiHeader->biCompression;
    m_WidthInBytes = WIDTHBYTES(m_XDim * m_BitsPerPixel);
}

// Image(HDIB hDIB)
// Constructor which creates an Image object, given the name of a DIB
// or BMP file.
// Image::Image(CString filename)
{
    CFile file;
    CFileException fe;
    BITMAPINFO *bmi_info;
    m_hpPackedData = NULL;

    if (!file.Open(filename, CFile::modeRead | CFile::shareDenyWrite, &fe))
    {
        CString msg("Error reading image file: ");
        msg += filename;
        MessageBox(NULL, msg, NULL, MB_ICONINFORMATION | MB_OK);
        m_fileOK = FALSE;
    }
    else
        m_fileOK = TRUE;

    // Try to read the DIB file, catch any exceptions.
    TRY
    {
        m_hDIB = ::ReadDIBFile(file);
    }
    CATCH(CFileException, eLoad)
    {
        file.Abort();
        MessageBox(NULL, "Error reading the image file", NULL,
            MB_ICONINFORMATION | MB_OK);
        m_hDIB = NULL;
        m_fileOK = FALSE;
    }
    END_CATCH

    m_lpDIB = (LPSTR) ::GlobalLock( (HGLOBAL) m_hDIB);

    // NOTE: THE FOLLOWING MEMBER POINTERS ARE ONLY VALID WHILE
    // WE KEEP THE DIB DATA LOCKED IN MEMORY. FOR THIS IMPLEMENTATION,
    // I LEAVE THE DATA LOCKED UNTIL THE OBJECT IS DESTROYED.

    bmi_info = (BITMAPINFO *) m_lpDIB;
    // Set up a pointer to the BITMAPINFOHEADER and RGBQUAD array.
    m_lpBmiHeader = &bmi_info->bmiHeader;
    m_lpBmiColors = &bmi_info->bmiColors[0];

    // Set the pointer to the image data.
    m_hpDIBits = (unsigned char *) ::FindDIBBits(m_lpDIB);

    m_BitsPerPixel = m_lpBmiHeader->biBitCount;
    m_XDim = m_lpBmiHeader->biWidth;
    m_YDim = m_lpBmiHeader->biHeight;
    m_Compression = m_lpBmiHeader->biCompression;
    m_WidthInBytes = WIDTHBYTES(m_XDim * m_BitsPerPixel);
}

// ~Image()
// The destructor for the Image class of objects.
// Image::~Image(void)
{
    ::GlobalUnlock( (HGLOBAL) m_hDIB);
    if (m_hpPackedData != NULL)
    {
        ::GlobalUnlock( (HGLOBAL) m_hPackedData);
        ::GlobalFree( (HGLOBAL) m_hPackedData);
    }
}

// *****
// FILE: Pft.h
// *****
// DESCRIPTION:
// Include file for Geoff's PFT routines. Callers of the PFT functions
// should include this header file to pick up the function prototypes.
// *****
// Copyright (C) Digimarc Corporation, 1996, all rights reserved.
// *****
void fft(float *ar, /* the real part of the array */
         float *ai, /* the imag part of the array */
         int nbits, /* log base 2 of the number of elements in the arrays */
         int inv, /* nonzero to indicate the inverse transform */
         float *wr, /* the real part of an array of coefficients */
         float *wi, /* the imag part of an array of coefficients */
         int neww; /* nonzero to indicate the coefficients must be calcd */)
{
    int fft2d(float *ar, float *ai, int nbits, int inv, float *wr, float *wi);
    void realfft_two_arrays(float *array1, float *array2,
                           int nbits, int inv, float *wr, float *wi, int neww);
    int realfft2d_in_place(float *ar, int nbits, int inv, float *wr, float *wi);
}

// *****
// FILE: Image.cpp
// *****
// Contains the implementation for the Image class. Image objects
// are used to contain the image data and provide a more convenient
// set of services related to accessing the image data as well as
// attribute variables describing the image.
// *****
#include "Image.h"
#include "dibapi.h"
#include "sdata.h"

// Image(HDIB hDIB)
// Constructor which creates an Image object, given a handle to
// a DIB which is already in memory.
// Image::Image(HDIB hDIB)
{
    BITMAPINFO *bmi_info;
    m_hpPackedData = NULL;
    m_fileOK = TRUE;
    m_hDIB = hDIB;

    m_lpDIB = (LPSTR) ::GlobalLock( (HGLOBAL) m_hDIB);

    // NOTE: THE FOLLOWING MEMBER POINTERS ARE ONLY VALID WHILE
    // WE KEEP THE DIB DATA LOCKED IN MEMORY. FOR THIS IMPLEMENTATION,
    // I LEAVE THE DATA LOCKED UNTIL THE OBJECT IS DESTROYED.

    bmi_info = (BITMAPINFO *) m_lpDIB;
    // Set up a pointer to the BITMAPINFOHEADER and RGBQUAD array.
    m_lpBmiHeader = &bmi_info->bmiHeader;
    m_lpBmiColors = &bmi_info->bmiColors[0];

    // Set the pointer to the image data.
    m_hpDIBits = (unsigned char *) ::FindDIBBits(m_lpDIB);
}

```

IMAG_BT.CPP

```

// *****
// FILE: Image.cpp
// *****
// Contains the implementation for the Image class. Image objects
// are used to contain the image data and provide a more convenient
// set of services related to accessing the image data as well as
// attribute variables describing the image.
// *****
#include "Image.h"
#include "dibapi.h"
#include "sdata.h"

// Image(HDIB hDIB)
// Constructor which creates an Image object, given a handle to
// a DIB which is already in memory.
// Image::Image(HDIB hDIB)
{
    BITMAPINFO *bmi_info;
    m_hpPackedData = NULL;
    m_fileOK = TRUE;
    m_hDIB = hDIB;

    m_lpDIB = (LPSTR) ::GlobalLock( (HGLOBAL) m_hDIB);

    // NOTE: THE FOLLOWING MEMBER POINTERS ARE ONLY VALID WHILE
    // WE KEEP THE DIB DATA LOCKED IN MEMORY. FOR THIS IMPLEMENTATION,
    // I LEAVE THE DATA LOCKED UNTIL THE OBJECT IS DESTROYED.

    bmi_info = (BITMAPINFO *) m_lpDIB;
    // Set up a pointer to the BITMAPINFOHEADER and RGBQUAD array.
    m_lpBmiHeader = &bmi_info->bmiHeader;
    m_lpBmiColors = &bmi_info->bmiColors[0];

    // Set the pointer to the image data.
    m_hpDIBits = (unsigned char *) ::FindDIBBits(m_lpDIB);
}

```



```

bmi_info = (BITMAPINFO *) m_lpDIB;
// Set up a pointer to the BITMAPINFOHEADER and RGBQUAD array.
m_lpBmiHeader = &bmi_info->bmiHeader;
m_lpBmiColors = &bmi_info->bmiColors[0]; // will be null for 24 bit

// Set the pointer to the image data.
m_hpDIBBits = (unsigned char *) ::FindDIBBits(m_lpDIB);

m_BitsPerPixel = m_lpBmiHeader->biBitCount;
m_XDim = m_lpBmiHeader->biWidth;
m_YDim = m_lpBmiHeader->biHeight;
m_Compression = m_lpBmiHeader->biCompression;
m_WidthInBytes = WIDTHBYTES(m_XDim * m_BitsPerPixel);

// Image(HDIB hDIB)
// Constructor which creates an Image object, given the name of a DIB
// or BMP file.
Image::Image(CString filename)
{
    CFile
    file;
    CFileException fe;
    BITMAPINFO *bmi_info;
    m_hpPackedData = NULL;

    if (!file.Open(filename, CFile::modeRead | CFile::shareDenyWrite, &fe))
    {
        CString msg("Error reading image file: ");
        msg += filename;
        MessageBox(NULL, msg, NULL, MB_ICONINFORMATION | MB_OK);
        m_fileOK = FALSE;
    }
    else
        m_fileOK = TRUE;

// Try to read the DIB file, catch any exceptions.
TRY
{
    m_hDIB = ::ReadDIBFile(file);
}
CATCH(CFileException, eLoad)
{
    file.Abort();
    MessageBox(NULL, "Error reading the image file", NULL,
        MB_ICONINFORMATION | MB_OK);
    m_hDIB = NULL;
    m_fileOK = FALSE;
}
END_CATCH

m_lpDIB = (LPSTR) ::GlobalLock( (HGLOBAL) m_hDIB);

// NOTE: THE FOLLOWING MEMBER POINTERS ARE ONLY VALID WHILE
// WE KEEP THE DIB DATA LOCKED IN MEMORY. FOR THIS IMPLEMENTATION,
// I LEAVE THE DATA LOCKED UNTIL THE OBJECT IS DESTROYED.

bmi_info = (BITMAPINFO *) m_lpDIB;
// Set up a pointer to the BITMAPINFOHEADER and RGBQUAD array.
m_lpBmiHeader = &bmi_info->bmiHeader;
m_lpBmiColors = &bmi_info->bmiColors[0];

// Set the pointer to the image data.
m_hpDIBBits = (unsigned char *) ::FindDIBBits(m_lpDIB);

m_BitsPerPixel = m_lpBmiHeader->biBitCount;
m_XDim = m_lpBmiHeader->biWidth;
m_YDim = m_lpBmiHeader->biHeight;
m_Compression = m_lpBmiHeader->biCompression;
m_WidthInBytes = WIDTHBYTES(m_XDim * m_BitsPerPixel);

// ~Image()
// The destructor for the Image class of objects.
Image::~Image(void)
{
    ::GlobalUnlock( (HGLOBAL) m_hDIB);

```

```

if (m_hpPackedData != NULL)
{
    ::GlobalUnlock( (HGLOBAL) m_hpPackedData);
    ::GlobalFree( (HGLOBAL) m_hpPackedData);
}

// MakePackedData()
// This function copies the DIB image data into a packed format. This
// is important for two reasons: 1) the DIB formatted data is arranged
// so that each scan line starts on a long word boundary, so there may
// be up to 3 unused bytes at the end of each scan line in the case of
// 8 bit data. This arrangement is inconvenient when passing the image
// data to the core algorithms. Also, 2) if a palette is being used
// (this is the case for all but 24 bit image data), this routine looks
// up the actual image values using the palette and places these values
// in the packed data array. The member variable m_hpPackedData is the
// handle to the packed data.

// The force_to_1_chan argument is an optional boolean. It defaults
// to FALSE (see function prototype in Image.h). If set to TRUE,
// only 1 channel of packed data is created, even if the image is 3
// channels. This is useful when creating snowy images from RGB
// images, since we currently always want 1 channel snowy images.
void Image::MakePackedData(BOOLEAN force_to_1_chan)
{
    unsigned char *hpline;
    unsigned char *hpData;
    int line_cnt, line, i, j;
    long size;
    BOOLEAN bottom_up;

    // Create space and get handle for the packed data of the image.
    size = m_XDim * m_YDim;
    // For 24 bit true color, we will pack R,G,B values, so triple the size.
    if (m_BitsPerPixel == 24 && force_to_1_chan == FALSE)
        size *= 3;
    m_hpPackedData = ::GlobalAlloc(GMEM_MOVEABLE | GMEM_ZEROINIT, size);
    if (m_hpPackedData == 0)
        AfxThrowMemoryException();

    // Lock the packed data global memory (leave locked until destructor).
    m_hpPackedData = (unsigned char *) ::GlobalLock( (HGLOBAL) m_hpPackedData);

    hpData = m_hpPackedData;

    // Image may be top to bottom or bottom to top.
    if (m_lpBmiHeader->biHeight > 0)
    {
        bottom_up = TRUE;
        line = m_YDim - 1;
    }
    else
    {
        bottom_up = FALSE;
        line = 0;
    }

    // TEST CODE
    // For Geoff: don't let it correct for bottom_up
    // bottom_up = FALSE;
    // line = 0;

    // Now go through each line and create the packed array.
    for (line_cnt = 0; line_cnt < m_YDim; line_cnt++)
    {
        // Set pointer to first byte for this scan line.
        hpline = &m_hpDIBBits[line * (long) m_WidthInBytes];
        for (i = 0, j = 0; i < m_XDim; i++)
        {
            if (m_BitsPerPixel == 24)
            {
                if (!force_to_1_chan)
                {
                    *hpData++ = hpline[j+2]; // red
                    *hpData++ = hpline[j+1]; // green
                    *hpData++ = hpline[j+0]; // blue
                }
                else
                {
                    *hpData++ = hpline[j+1]; // take just green to convert
                    // to 1 channel data.
                }
                j += 3;
            }
            else
            {

```

```

    {
        // For 8 bit (and any other non 24 bit data) we
        // take the image data to be indices into the color
        // table. We look up the actual value. Note we
        // assume grey-scale (i.e., r,g,b triples are all equal -
        // we read the green.
        *hpData++ = m_lpBmiColors(hpline[i]).rgbGreen;
    }
    if (bottom_up) line--;
    else line++;
}

// UnpackData()
// This function moves the contents of the packed data array back into
// the DIB data space. This would be used, for example, after one the
// core algorithms have been used to sign the data in the packed array,
// and we want to update the DIB to reflect the changes. Note that this
// requires that we create our own palette, since otherwise we don't know
// that the new data values have corresponding entries in the palette.

// WARNING: CURRENT IMPLEMENTATION ASSUMES 8 BIT GRAY-SCALE IMAGE DATA
// OR 24 BIT COLOR IMAGE DATA
// void Image::UnpackData(void)
// {
//     unsigned char *hpline;
//     unsigned char *hpData;
//     int line_cnt, line, i, j;
//     BOOLEAN bottom_up;
//
//     // Image may be top to bottom or bottom to top.
//     if (m_lpBmiHeader->biHeight > 0)
//     {
//         bottom_up = TRUE;
//         line = m_YDim - 1;
//     }
//     else
//     {
//         bottom_up = FALSE;
//         line = 0;
//     }
//
//     // TEST CODE
//     // For Geoff: don't let it correct for bottom_up
//     // bottom_up = FALSE;
//     // line = 0;
//
//     hpData = m_hpPackedData;
//     for (line_cnt = 0; line_cnt < m_YDim; line_cnt++)
//     {
//         // Set pointer to first byte for this scan line.
//         hpline = &m_hpPackedData[line * (long) m_WidthInBytes];
//         for (i = 0, j = 0; i < m_XDim; i++)
//         {
//             if (m_BitsPerPixel == 24)
//             {
//                 hpline[j+2] = *hpData++; // red
//                 hpline[j+1] = *hpData++; // green
//                 hpline[j] = *hpData++; // blue
//                 j += 3;
//             }
//             else
//             {
//                 hpline[i] = *hpData++;
//             }
//             if (bottom_up) line--;
//             else line++;
//         }
//     }
//
//     // Next, we force the palette to be our standard 8 bit grey-scale
//     // palette.
//     if (m_BitsPerPixel == 8)
//     {
//         // Set ptr to beginning of palette
//         LPKQBQUAD pal = m_lpBmiColors;
//         for (i = 0; i < 256; i++)
//         {
//             pal[i].rgbBlue = pal[i].rgbGreen = pal[i].rgbRed = i;
//         }
//     }
//     else if (m_BitsPerPixel == 24)
//     {
//         // Don't do any palette work for 24 bit color: there is no palette.
//     }
//     else

```

```

    {
        MessageBox(NULL, "Can only unpack 8 and 24 bit image data", NULL,
            MB_ICONEXCLAMATION | MB_OK);
    }
}

//***** IMAGE_H *****
// FILE: Image.h
//
// * DESCRIPTION:
// * The Image class is used to read BMP and DIB image files, and
// * manage an internal representation of them in memory. The goal is
// * to provide a set of service which insulate the caller from having to
// * deal with the specifics of the DIB format. Also, the approach tends
// * to isolate platform specific and file format specific details to this
// * class. For example, adding support for a different type of file
// * format would affect this class, but not the callers.
// * This header file should be included by any module which creates or
// * makes use of Image objects.
//
// * CREATION DATE: September 5, 1995
//
// * Copyright (c) 1995 Digimarc Incorporated. All rights reserved.
// *****
// #ifndef IMAGE_H
// #define IMAGE_H
//
// #include "stdafx.h"
// #include "dibapi.h"
//
// class Image
// {
// public:
//     // Constructors...
//     Image(HDIB hDIB); // Takes a handle to a loaded DIB
//     Image(CString filename); // Takes a filename
//     ~Image(void);
//     void Image::MakePackedData(void);
//     void Image::MakePackedData(BOOLEAN force_to_1_chan = FALSE);
//     void Image::UnpackData();
//
//     // Accessors:
//     HDIB GetHDIB(void) {return m_hDIB;}
//     LPSTR GetLPSTR(void) {return m_lpDIB;}
//     BITMAPINFOHEADER *GetBmiHdr(void) {return m_lpBmiHeader;}
//     RGBQUAD *GetPalette(void) {return m_lpBmiColors;}
//     unsigned char *GetPackedData(void) {return m_hpPackedData;}
//     int GetBitsPerPixel(void) {return m_BitsPerPixel;}
//     WORD GetSizeOfPixel(void) {return m_SizeOfPixel;}
//     WORD GetSizeOfPalette(void) {return m_SizeOfPalette;}
//     WORD GetSizeOfHeader(void) {return m_SizeOfHeader;}
//     LONG GetNumColors(void) {return m_DIBNumColors(m_lpDIB);}
//     LONG GetXDim(void) {return m_XDim;}
//     LONG GetYDim(void) {return m_YDim;}
//     BOOL GetFileOK(void) {return m_fileOK;}
//
//     // Private member functions
//     // Private data
// private:
//     // Handle to the DIB.
//     HDIB m_hDIB;
//     LPSTR m_lpDIB; // Pointer to top of DIB, locked in memory
//     // Pointers to the bitmap info header structure, and the palette array.
//     BITMAPINFOHEADER *m_lpBmiHeader; // Points to header structure
//     RGBQUAD *m_lpBmiColors; // Pts to beginning of palette array
//     unsigned char *m_hpDIBBits; // Pointer to DIB bits
//     HANDLE m_hpPackedData; // Handle for the packed data space
//     unsigned char *m_hpPackedData; // Pointer to packed copy of data.
//     LONG m_XDim; // X dimension of image
//     LONG m_YDim; // Y dimension of image (number of lines)
//     int m_BitsPerPixel;
//     int m_WidthInBytes;
//     LONG m_Compression;
//     DWORD m_fileOK;
// };

```



```

bmhdr.bfType = DIB_HEADER_MARKER; // "BM"

// Calculating the size of the DIB is a bit tricky (if we want to
// do it right). The easiest way to do this is to call GlobalSize()
// on our global handle, but since the size of our global memory may have
// been padded a few bytes, we may end up writing out a few too
// many bytes to the file (which may cause problems with some apps).
// So, instead let's calculate the size manually (if we can)
// First, find size of header plus size of color table. Since the
// first DWORD in both BITMAPINFOHEADER and BITMAPCOREHEADER contains
// the size of the structure, let's use this.

dwdIBSize = *(LPDWORD)lpBI + ::PaLETTESize((LPSTR)lpBI); // Partial Calculation
// Now calculate the size of the image
if ((lpBI->biCompression == BI_RGB) || (lpBI->biCompression == BI_RGB4))
{
    // It's an RGB bitmap, we can't calculate size, so trust the
    // biSizeImage field
    dwDIBSize += lpBI->biSizeImage;
}
else
{
    DWORD dwBmBitsSize; // Size of Bitmap Bits only
    // It's not RGB, so size is Width (DWORD aligned) * Height
    dwBmBitsSize = WIDTHBYTES((lpBI->biWidth)*((DWORD)lpBI->biBitCount)) * lpBI->biHeight;
    dwDIBSize += dwBmBitsSize;
    // Now, since we have calculated the correct size, why don't we
    // fill in the biSizeImage field (this will fix any .BMP files which
    // have this field incorrect).
    lpBI->biSizeImage = dwBmBitsSize;
}

// Calculate the file size by adding the DIB size to sizeof(BITMAPFILEHEADER)
bmhdr.bfSize = dwDIBSize + sizeof(BITMAPFILEHEADER);
bmhdr.bfReserved1 = 0;
bmhdr.bfReserved2 = 0;

/*
 * Now, calculate the offset the actual bitmap bits will be in
 * the file -. It's the Bitmap file header plus the DIB header,
 * plus the size of the color table.
bmhdr.bfOffBits = (DWORD)sizeof(BITMAPFILEHEADER) + lpBI->biSize
                + PaLETTESize((LPSTR)lpBI);
*/

TRY
{
    // Write the file header
    file.Write((LPSTR)&bmhdr, sizeof(BITMAPFILEHEADER));
    // Write the DIB header and the bits
    file.Write((LPSTR)&lpBI, dwDIBSize);
}
CATCH (CFileException, e)
{
    ::GlobalUnlock((HGLOBAL) hDib);
    THROW_LAST();
}
END_CATCH

::GlobalUnlock((HGLOBAL) hDib);
return TRUE;
}

/*****
Function: ReadDIBFile (CFiles)
Purpose: Reads in the specified DIB file into a global chunk of
memory.
Returns: A handle to a dib (hDIB) if successful.
        NULL if an error occurs.
Comments: BITMAPFILEHEADER is stripped off of the DIB. Everything
from the end of the BITMAPFILEHEADER structure on is
*****/

```

```

        returned in the global memory handle.
*****/
HDIB WINAPI ReadDIBFile(CFile& file)
{
    BITMAPFILEHEADER bmfHeader;
    DWORD dwBmBitsSize;
    HDIB hDIB;
    LPSTR pDIB;
    /*
     * get length of DIB in bytes for use when reading
     */
    dwBmBitsSize = file.GetLength();
    /*
     * Go read the DIB file header and check if it's valid.
     */
    if ((file.Read((LPSTR)&bmfHeader, sizeof(bmfHeader)) !=
        sizeof(bmfHeader)) || (bmfHeader.bfType != DIB_HEADER_MARKER))
    {
        return NULL;
    }
    /*
     * Allocate memory for DIB
     */
    hDIB = (HDIB) ::GlobalAlloc(GMEM_MOVEABLE | GMEM_ZEROINIT, dwBmBitsSize);
    if (hDIB == 0)
    {
        return NULL;
    }
    pDIB = (LPSTR) ::GlobalLock((HGLOBAL) hDIB);
    /*
     * Go read the bits.
     */
    if (file.Read(huge(pDIB, dwBmBitsSize - sizeof(BITMAPFILEHEADER)) !=
        dwBmBitsSize - sizeof(BITMAPFILEHEADER)) )
    {
        ::GlobalUnlock((HGLOBAL) hDIB);
        ::GlobalFree((HGLOBAL) hDIB);
        return NULL;
    }
    ::GlobalUnlock((HGLOBAL) hDIB);
    return hDIB;
}

/*****
 * FILE: PackMsg.cpp
 *
 * DESCRIPTION:
 * The PackedMsg class is responsible for creating an efficient binary*
 * coding representation of the ASCII message the user wishes to embed*
 * in the image. This representation is "efficient" in that it packs*
 * the message into a format which requires fewer total bits than that*
 * used by the equivalent ASCII representation.*
 * Currently, the packing scheme translates each ASCII character of the*
 * user message to a value which can be represented with 6 bits. Some*
 * ASCII characters have no representation, of course, since only 64*
 * alphanumeric and special characters can be represented by the 6 bit*
 * code. See the enumeration in the Packmsg.h file for the exact*
 * translations used.
 * This C++ file contains the implementation code for the class.*
 *
 * CREATION DATE: August 31, 1995
 *
 * Copyright (c) 1995 Digimarc Incorporated, all rights reserved.*
 *****/
#include "stdafx.h"
#include "packmsg.h"
#include <string.h>
#include <ctype.h>

typedef char * Compact_Msg;

/*****
// PackedMsg(const char *user_msg)
//
// This is the PackedMsg constructor which is given an ASCII
*****/

```



```

// message for use by the signer. It creates an array of
// packed characters (a more compact representation than
// ASCII), computes the checksum for the compact string,
// and then creates a bit array containing the compact
// message (this is the form the signer core algorithms
// require).
// PackedMsg::PackedMsg(const char *user_msg)
{
    m_correctBits = 0;
    m_checksum = 0;
    m_computedReaderChecksum = 0;
    m_recoveredReaderChecksum = 0;

    // Save the length, and a copy of the original user (ascii) message.
    m_msgLength = strlen(user_msg);
    m_asciiMsg = new char[m_msgLength+1];
    strcpy(m_asciiMsg, user_msg);
    m_recoveredAsciiMsg = new char[m_msgLength+1];

    // Note it is null terminated.

    // Allocate space for the packed message. Note there's no NULL termination.
    m_compactMsg = new char[m_msgLength];

    // Call the function which translates to compact form.
    PackMessage();

    // Compute the checksum of the compact message string
    m_checksum = ComputeChecksum(m_compactMsg, m_msgLength);

    // Allocate space for the MsgBitArray, which puts one bit of the
    // packed message in each char of an unsigned char array (this is
    // the format that the current core signer needs.
    // Also, we include space for checksum of same length as 1 char.
    // Also allocate space for the ReaderBitArray, which reader will use.
    m_msgBitArrayLength = (m_msgLength+1) * PACKED_BITS_PER_CHAR;
    m_msgBitArray = new unsigned char[m_msgBitArrayLength];
    m_readerBitArray = new unsigned char[m_msgBitArrayLength];

    unsigned char *p_bit_array = m_msgBitArray;
    unsigned char *p_reader_array = m_readerBitArray;
    int i, j;
    unsigned char mask;
    for (i = 0; i < m_msgLength; i++)
    {
        for (j = PACKED_BITS_PER_CHAR - 1; j >= 0; j--)
        {
            mask = 1 << j;
            if (m_compactMsg[i] & mask)
                *p_bit_array = 1;
            else
                *p_bit_array = 0;

            *p_bit_array++;
            *p_reader_array++ = 0;
        }
        // clear the readers array.
    }

    // Continue by putting the checksum in the final PACKED_BITS_PER_CHAR
    // elements of the bit array
    for (j = PACKED_BITS_PER_CHAR - 1; j >= 0; j--)
    {
        mask = 1 << j;
        if (m_checksum & mask)
            *p_bit_array = 1;
        else
            *p_bit_array = 0;

        *p_bit_array++;
        *p_reader_array++ = 0;
    }
    // clear the readers array.
}

// The PackedMsg constructor which is the length of a message to be read.
PackedMsg::PackedMsg(int msg_length)
{
    int i;

    m_correctBits = 0;

    // Save the length, and allocate space for the ASCII message.
    m_msgLength = msg_length;
    m_asciiMsg = new char[m_msgLength+1];

    // Null out the ascii storage
    for (i = 0; i < m_msgLength+1; i++)
        m_asciiMsg[i] = '\0';

    // Allocate space for the packed message. Note there's no NULL termination.
    m_compactMsg = new char[m_msgLength];

```

```

// Allocate space for the MsgBitArray, which will hold one bit of the
// packed message in each char of an unsigned char array (this is
// the format that the current core signer needs.
// Also, we include space for checksum of same length as 1 char.
// Also allocate space for the ReaderBitArray, which reader will use.
m_msgBitArrayLength = (m_msgLength+1) * PACKED_BITS_PER_CHAR;
m_msgBitArray = new unsigned char[m_msgBitArrayLength];
m_readerBitArray = new unsigned char[m_msgBitArrayLength];

// The Destructor
PackedMsg::~PackedMsg()
{
    delete [] m_asciiMsg;
    delete [] m_compactMsg;
    delete [] m_msgBitArray;
    delete [] m_readerBitArray;
    delete [] m_recoveredAsciiMsg;
}

// Converts the ASCII message into an array of "packed" char-
// acters (currently 6 bits per packed character) which require
// a minimum of bandwidth in the Digimarc signed image.
// void PackMessage(void)
{
    int i;
    char ascii_ch;
    for (i = 0; i < m_msgLength; i++)
    {
        ascii_ch = toupper(m_asciiMsg[i]);
        if (ascii_ch >= '0' && ascii_ch <= '9')
            m_compactMsg[i] = zero + (ascii_ch - '0');
        else if (ascii_ch >= 'A' && ascii_ch <= 'Z')
        {
            m_compactMsg[i] = A + (ascii_ch - 'A');
        }
    }

    // Check for special characters and encode them.
    else switch (ascii_ch)
    {
        case ' ': m_compactMsg[i] = space;
        break;
        case '.': m_compactMsg[i] = period;
        break;
        case ',': m_compactMsg[i] = comma;
        break;
        case ':': m_compactMsg[i] = colon;
        break;
        case '/': m_compactMsg[i] = slash;
        break;
        case '\\': m_compactMsg[i] = backslash;
        break;
        default:
            m_compactMsg[i] = undefined;
            // Warn user that an undefined character was found.
            CString warn_msg;
            warn_msg = "Sorry, but \"";
            warn_msg += CString(ascii_ch);
            warn_msg += "\" is not part of the Digimarc character set.";
            warn_msg += "\nUnit will be replaced by a '?'. ";
            MessageBox(NULL, warn_msg, "Warning", MB_ICONINFORMATION | MB_OK);
            break;
    }
}

// BitsToString()
// Function which reads the recovered bit array, containing one bit of
// the packed binary message in each char element, and packs these bits
// into the m_compactMsg array (which then contains one packed msg
// character per element). It then converts the compactMsg to
// ASCII and puts the resulting characters in the m_recoveredAsciiMsg
// array. Also, the last PACKED_BITS_PER_CHAR bits contain the checksum.
// This is recovered and stored in the m_recoveredChecksum variable.
// void BitsToString(void)

```

```

    // Compute the checksum of the read message
    m_computedReaderChecksum = ComputeChecksum(m_compactMsg, m_msgLength);
}

// First, build the m_compactMsg array from the m_readerBitArray.

//bit array_ptr = m_readerBitArray;
p_read_bits = m_readerBitArray;
p_signed_bits = m_msgBitArray;
m_correctBits = 0;
for (i = 0; i < m_msgLength; i++)
{
    m_compactMsg[i] = 0; // Start with nothing.
    for (j = PACKED_BITS_PER_CHAR - 1; j >= 0; j--)
    {
        if (*p_read_bits == 1)
        {
            bit = 1;
            m_compactMsg[i] |= (bit << j);
        }
        // Compute bit success rate metric:
        if (*p_read_bits == *p_signed_bits)
            m_correctBits++;
        p_read_bits++;
        p_signed_bits++;
    }
}

// Now recover the checksum from the end of the bit array.
m_recoveredChecksum = 0;
for (j = PACKED_BITS_PER_CHAR - 1; j >= 0; j--)
{
    if (*p_read_bits == 1)
    {
        m_recoveredChecksum |= (1 << j);
    }
    // Compute bit success rate metric:
    if (*p_read_bits == *p_signed_bits)
        m_correctBits++;
    p_read_bits++;
    p_signed_bits++;
}

// Next, convert the compact form to an ASCII string.
for (i = 0; i < m_msgLength; i++)
{
    if (m_compactMsg[i] >= zero && m_compactMsg[i] <= nine)
        m_recoveredAsciiMsg[i] = '0' + m_compactMsg[i] - zero;
    else if (m_compactMsg[i] >= A && m_compactMsg[i] <= Z)
        m_recoveredAsciiMsg[i] = 'A' + m_compactMsg[i] - A;
    else switch (m_compactMsg[i])
    {
        case space:
            m_recoveredAsciiMsg[i] = ' ';
            break;
        case period:
            m_recoveredAsciiMsg[i] = '.';
            break;
        case comma:
            m_recoveredAsciiMsg[i] = ',';
            break;
        case colon:
            m_recoveredAsciiMsg[i] = ':';
            break;
        case slash:
            m_recoveredAsciiMsg[i] = '/';
            break;
        case backslash:
            m_recoveredAsciiMsg[i] = '\\';
            break;
        default:
            m_recoveredAsciiMsg[i] = '?'; // When we don't recognize the character.
            break;
    }
}
// Add a Null terminator
m_recoveredAsciiMsg[m_msgLength] = '\0';

```

```

// Compute the checksum of the read message
m_computedReaderChecksum = ComputeChecksum(m_compactMsg, m_msgLength);
}

// ComputeChecksum()
// This function is passed a pointer to the compact message
// string containing a message. It computes and returns the checksum.
// The checksum algorithm used is a simple "spiral add", and the
// size of the checksum is PACKED_BITS_PER_CHAR (although it is
// stored as an unsigned char).
// NOTE:
// There is an implicit assumption that PACKED_BITS_PER_CHAR < 8
// If this changes, mods will be needed in this code.
// unsigned char PackedMsg::ComputeChecksum(char *pMsg, int length)
{
    int i;
    unsigned char csum = 0;
    const unsigned char carry_bit_mask = (1 << PACKED_BITS_PER_CHAR);
    const unsigned char remove_carry_bit_mask = ~carry_bit_mask;
    for (i = 0; i < length; i++)
    {
        // Rotate the checksum: shift left and OR in the carry bit.
        csum = csum << 1;
        if (csum & carry_bit_mask)
        {
            csum |= 1;
            csum &= remove_carry_bit_mask;
        }
        // Add the next character
        csum += (unsigned char) *pMsg;
        // We want an unsigned add of length PACKED_BITS_PER_CHAR,
        // so remove the carry bit if its there.
        csum &= remove_carry_bit_mask;
        pMsg++;
    }
    return csum;
}

//*****
// FILE: PackMsg.h
//
// DESCRIPTION:
// The PackMsg class is responsible for creating an efficient binary*
// coding representation of the ASCII message the user wishes to embed*
// in the image. This representation is "efficient" in that it packs*
// the message into a format which requires fewer total bits than that*
// used by the equivalent ASCII representation.*
//
// This header file should be included by any module which creates or*
// makes use of PackMsg objects.*
//
// CREATION DATE: August 16, 1995
//
// Copyright (c) 1995 Digimarc Incorporated, all rights reserved.*
//*****
#ifndef PACKMSG_H
#define PACKMSG_H

#include "digimarc.h"
#include "Params.h"

#define PACKED_BITS_PER_CHAR 6 // We will use 6 bits per user character

// We're going to use a 6 bit representation of up to 64 alphanumeric
// plus special characters. The following enumeration indicates how
// each will be represented. There first item takes value 0, 2nd item
// takes 1, ...
enum PackedChar
{
    zero, one, two, three, four, five, six, seven, eight, nine,
    A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z,
    space, period, comma, colon, slash, backslash,
    undefined
};

```

```

typedef char * Compact_Msg;

class PackedMsg
{
// Public member functions
public:
// Constructor: takes user's input message and creates the packed version.
PackedMsg(const char *user_msg);

// A Constructor for use by the reader.
PackedMsg(int msg_length);

// An accessor allows callers read-only access to the packed msg.
const Compact_Msg getCompactMsg(void) const;
int getCompactMsgSize(void) const;
unsigned char *getMsgBitArray(void) const {return m_msgBitArray;}
int getMsgBitArrayLength(void) const {return m_msgBitArrayLength;}
char *getAsciiMsg(void) const {return m_asciiMsg;}
unsigned char *getReaderBitArray(void) const {return m_readerBitArray;}
char *getRecoveredAsciiMsg(void) const {return m_recoveredAsciiMsg;}

int GetNumCorrectBits(void) const {return m_correctBits;}
float GetPercentCorrect(void) const
{return (float) m_correctBits * (float)100.0 / (float) m_msgBitArrayLength;}

// Checksum accessors.
unsigned char GetSignerChecksum(void) {return m_checksum;}
unsigned char GetReaderChecksum(void) {return m_recoveredChecksum;}
unsigned char GetComputedReaderChecksum(void) {return m_computedReaderChecksum;}

int GetMsgLength(void) const {return m_msgLength;}

// Function to unpack a message, for use by the recognizer...
void BitToString(void);

// Destructor
~PackedMsg(void);

// Private member functions
private:
void PackMessage(void);
unsigned char ComputeChecksum(char *pMsg, int length);

// Private data
private:
char *m_asciiMsg; // The original ASCII message ASCII (null terminated).
int m_msgLength; // No. of chars (not included null terminator).
Compact_Msg m_compactMsg; // The message in the packed format.

unsigned char *m_msgBitArray; // Core signer algorithm wants one bit per char.
int m_msgBitArrayLength; // Includes checksum.
unsigned char *m_readerBitArray; // Array of bits recovered by reader,
char *m_recoveredAsciiMsg; // The recovered message

unsigned char m_checksum;
unsigned char m_recoveredChecksum;
unsigned char m_computedReaderChecksum;

int m_correctBits;
};

#endif // PACKMSG_H

/*=====
* FILE: Params.cpp
* DESCRIPTION:
* Implementation of the Parameters classes: SignerParams and
* ReaderParams.
*
*
*
* CREATION DATE: September 8, 1995
* Copyright (c) 1995 Digimarc Incorporated, all rights reserved.
* =====*/
#include "params.h"
#include "stdafx.h"
#include <string.h>
#include <strstream>

// =====
// SIGNERPARAMS.CPP
// =====
// Constructor based on command line
SignerParams::SignerParams(LPSTR cmd_line)
{
char *dash_ptr, *cmd_type, *cmd, *commands;
const char *dbg_msg_ptr;

parameters.input_filename = NULL;
parameters.message = "Default Message";
parameters.output_filename = NULL;
parameters.registry_name = NULL;
parameters.user_key = 1;
parameters.gain = (float) 100.0;
parameters.gamma = (float) 0.07;
parameters.bump_size = 1;
parameters.lut_scale = (float) 100.0;
parameters.super_reader_flag = FALSE;
dbg_msg_ptr = (const char *) GetMessage();
TRACE("Debug in SignerParams constructor. Message is: %s\n", dbg_msg_ptr);

// Make a copy of the command line that we can mutilate
commands = new char[strlen(cmd_line) + 1];
strcpy(commands, cmd_line);

dash_ptr = NULL;

// If the command line doesn't start w/ a '-', then the command line is
// a single argument: the filename. This case comes up when the program
// is invoked by dragging a filename onto the executable in Win95 explorer.
if (strlen(cmd_line) > 0 && cmd_line[0] != '-')
{
parameters.input_filename = new char[strlen(cmd_line) + 1];
strcpy(parameters.input_filename, cmd_line);
}

// Otherwise, we check for the multiple argument format of the command line,
// in which arguments pairs are used, e.g., "-f <filename>".
else
{
do
{
// Find the last '-' character
dash_ptr = strrchr(cmd_line, '-');
if (dash_ptr != NULL)
{
cmd_type = dash_ptr + 1;
cmd = cmd_type + 1;

// Create an in-core input stream
istream inStream(cmd, strlen(cmd));

switch (*cmd_type)
{
case 'g':
case 'G':
inStream >> parameters.gain;
break;
case 'f':
case 'F':
parameters.input_filename = new char[strlen(cmd) + 1];
inStream >> parameters.input_filename;
break;
case 'm':
case 'M':
// parameters.message = new char[strlen(cmd) + 1];
// inStream.getline(parameters.message,
// strlen(cmd)+1,
// '\0');
// parameters.message = cmd;
break;
case 'z':
case 'Z':
inStream >> parameters.gamma;
break;
default:
// Lop off the last argument by replacing the dash with a NULL;
*dash_ptr = '\0';
} while (dash_ptr != NULL);
}
}
}
}

```



```
// "Super user" inputs, useful for testing and tuning, go here.
```

```
// Non user inputs will go here...
```

```
reader_param_struct;
```

```
class ReaderParams
```

```
{ // Public member functions and data structures
```

```
public:
```

```
ReaderParams(int argc, char *argv[]); // Constructor for non-gui (cmd line) version
```

```
// Create an accessor which returns a ptr to a const copy of the parameters structure.
```

```
// An alternative is to write accessors for each individual parameter.
```

```
const reader_param_struct * getParams(void) const;
```

```
// Private member functions and data structures
```

```
private:
```

```
reader_param_struct parameters; // structure containing the user parameters.
```

```
// Function which warns user if parameters are not all present or look incorrect.
```

```
// It will also throw an exception if things are not right.
```

```
checkParams(void);
```

```
};
```

```
#endif // PARAMS_H
```

PARAMSDLG.CPP

```
// parmsdlg.cpp : implementation file
```

```
//
```

```
#include "stdafx.h"
```

```
#include "signer.h"
```

```
#include "parmsdlg.h"
```

```
#ifdef _DEBUG
```

```
#undef THIS_FILE
```

```
static char BASED_CODE THIS_FILE[] = __FILE__;
```

```
#endif
```

```
////////////////////////////////////
```

```
ParmsDlg dialog
```

```
ParmsDlg::ParmsDlg(CWnd* pParent /*=NULL*/) :
```

```
{ : CDialog(ParmsDlg::IDD, pParent)
```

```
{ //{{AFX_DATA_INIT(ParmsDlg)
```

```
m_message = _T("");
```

```
m_gain_from_edit_box = (float) 0.0;
```

```
m_key = 0;
```

```
m_bump_size = 0;
```

```
m_detail_lut_scale = 0.0f;
```

```
//}}AFX_DATA_INIT
```

```
}
```

```
void ParmsDlg::DoDataExchange(CDataExchange* pDX)
```

```
{ CDialog::DoDataExchange(pDX);
```

```
//{{AFX_DATA_MAP(ParmsDlg)
```

```
DDX_Text(pDX, IDC_MESSAGE, m_message);
```

```
DDV_MaxChars(pDX, m_message, 256);
```

```
DDX_Text(pDX, IDC_EDIT_GAIN, m_gain_from_edit_box);
```

```
DDV_MinMaxFloat(pDX, m_gain_from_edit_box, 1.e-003f, 1.e+006f);
```

```
DDX_Text(pDX, IDC_EDIT_KEY, m_key);
```

```
DDV_Text(pDX, IDC_BUMP_SIZE, m_bump_size);
```

```
DDV_MinMaxInt(pDX, m_bump_size, 1, 256);
```

```
DDX_Text(pDX, IDC_DETAIL_SCALE, m_detail_lut_scale);
```

```
DDV_MinMaxFloat(pDX, m_detail_lut_scale, 1.e-003f, 1.e+006f);
```

```
//}}AFX_DATA_MAP
```

```
}
```

```
BEGIN MESSAGE MAP(ParmsDlg, CDialog)
```

```
//{{AFX_MSG_MAP(ParmsDlg)
```

```
ON_COMMAND(ID_SETTINGS_SIGNER, OnSettingsSigner)
```

```
//}}AFX_MSG_MAP
```

```
END_MESSAGE_MAP()
```

```
////////////////////////////////////
```

```
// ParmsDlg message handlers  
void ParmsDlg::OnOK()  
{  
    CDialog::OnOK();  
}  
  
void ParmsDlg::OnSettingsSigner()  
{  
    // TODO: Add your command handler code here  
}
```

PARAMSDLG.H

```
// parmsdlg.h : header file  
//  
#include "stdafx.h"  
////////////////////////////////////  
// ParmsDlg dialog
```

```
class ParmsDlg : public CDialog
```

```
{ // Construction
```

```
public:
```

```
ParmsDlg(CWnd* pParent = NULL); // standard constructor
```

```
// Dialog Data
```

```
//{{AFX_DATA(ParmsDlg)
```

```
enum { IDD = IDD_PARAMS_DIALOG };
```

```
CString m_message;
```

```
float m_gain_from_edit_box;
```

```
UINT m_key;
```

```
int m_bump_size;
```

```
float m_detail_lut_scale;
```

```
//}}AFX_DATA
```

```
// Implementation
```

```
protected:
```

```
virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
```

```
// Generated message map functions
```

```
//{{AFX_MSG(ParmsDlg)
```

```
virtual void OnOK();
```

```
afx_msg void OnSettingsSigner();
```

```
//}}AFX_MSG
```

```
DECLARE_MESSAGE_MAP()
```

```
};
```

RAWIMAGE.H

```
////////////////////////////////////
```

```
* FILE: RawImage.h
```

```
* *
```

```
* DESCRIPTION:
```

```
* RawImage objects are used to convert images from popular formats *
```

```
* to the raw image format used internally by the Digimarc system. *
```

```
* Typically, the RawImage constructor is given an input file as an *
```

```
* argument, and the constructor is responsible for reading the file. *
```

```
* and performing the necessary operations to convert it into the raw *
```

```
* format.
```

```
* RawImage objects also are able to perform the inverse conversion, *
```

```
* creating image files in various standard formats from the internal *
```

```
* raw representation.
```

```
* *
```

```
* The initial implementation will only except TIFF files as inputs, *
```

```
* and will make use of the public domain software LibTiff in order *
```

```
* to read and write TIFF files.
```

```
* *
```

```
* This header file should be included by any module which creates or *
```

```
* makes use of RawImage objects.
```

```
* *
```

```
* CREATION DATE: August 15, 1995
```

```
* *
```

```
* Copyright (c) 1995 Digimarc Incorporated, all rights reserved. *
```

```
\\
```

```
#ifndef RAWIMAGE_H
```

```
#define RAWIMAGE_H
```

```
#include "digimarc.h"
```

```
#include "Params.h"
```

```
// Since the exact internal representation may change, use a typedef.
```

```

// This will allow a single change to modify all references to the
// raw image data format.
// Also note that in the future we will need several raw image representation.
typedef long * Raw_Data;

class RawImage
{
// Public member functions and data structures
public:
    RawImage(SignerParams *params);

    // Member function which gives caller access to the raw image and its attributes.
    const int getxdim(void);
    const int getydim(void);

    // This accessor returns a const pointer to a read-only image.
    const Raw_Data getImage(void) const;

    // This accessor returns a const pointer to a writable image.
    Raw_Data * getWritableImage(void) const;

    // Member function used to convert the raw image to an output TIFF file.
    writeTiff(char *filename);

    // Private data. Users of rawImage objects get at these through accessors only.
private:
    int xdim; // X dimension of image
    int ydim; // Y dimension of image
    Raw_Data image; // Ptr to array of image data
};

//endif // RAWIMAGE_H

//////////////////////////////////////
// FILE: Read.cpp
//////////////////////////////////////

// DESCRIPTION:
// Core recognition functions of the Digimarc technology
// Created August 1995
// This particular code uses "raster" based processing as opposed to 2D based
// Copyright (C) 1996 Digimarc Corporation, all rights reserved.
//include "read.h"
//include "sign.h"
//include "stdafx.h"
//include <math.h>

/* Constants */
const float epsilon = (float) 0.000001;

//////////////////////////////////////
// read_8bit_single_channel_or_color()
//////////////////////////////////////
// Used to read (or "recognize") the embedded digimarc signature in
// either a gray-scale or color image. Set number_channels to 1 for
// gray-scale, 3 for color.
//////////////////////////////////////
int read_8bit_single_channel_or_color(
    unsigned char *data, // input data to be recognized */
    long original_xdim, // it's x dimension */
    long original_ydim, // it's y dimension */
    long x_offset, // x offset of segment */
    long y_offset, // y offset of segment */
    long x_extent, // x extent of segment */
    long y_extent, // y extent of segment */
    int message_length, // length of message in BITS, also length of message
    unsigned char *key, // original 8 bit random key */
    long key_length, // key_length often equal to data_length but not always */
    char *key_lut, // look up table mapping key value */
    float *luminance_lut, // look up table mapping the signature level to
    float *detail_lut, // look up table mapping the signature level to
    luminance*/

    unsigned char *thumbnail, // if available, use pointer, otherwise NULL */
    unsigned char *original_data, // if available, use pointer, otherwise NULL */

    const unsigned char *referenceBitArray, // bit array ptr: either the known message or
    estimate, // we will compute a return a crude metric indicating
    confidence.

    float *metric,
    float *range,
    unsigned char *message,
    int number_channels,
    int bumps
) {
    unsigned char *pkey, *pdata;
    long i, line, bit;
    int temp, status=1;
    float *key_value = new float[x_extent];
    float *data_float = new float[x_extent];
    float *orig_float = new float[x_extent];
    float *bit_total = new float[message_length];
    //float *bit_mag = new float[message_length];
    float *pkey_value, *pdata_float;
    float filter_cf = (float)0.5; // kludge for now
    double maxdiff = 40.0; // kludge for now

    int key_xlength = 1+(original_xdim-1)/bumps;
    for(i=0; i<message_length; i++)
    {
        bit_total[i] = (float) 0.0;
        //bit_mag[i] = (float) 0.0;
    }

    pdata = data;
    for(line=y_offset; line<(y_offset+y_extent); line++)

```

```

float *range,
    unsigned char *message,
    int number_channels,
    int reading_mode,
    int bumps
) {
    int status = 1;

    if(reading_mode == 0){
        read_8bit_single_channel_OLD_plus_color(
            data, original_xdim, original_ydim, x_offset, y_offset,
            x_extent, y_extent, message_length, key, key_length, key_lut,
            luminance_lut, detail_lut, thumbnail, original_data, referenceBitArray,
            metric, range, message, number_channels, bumps);
    }
    else if(reading_mode == 1){
        read super(
            data, original_xdim, original_ydim, x_offset, y_offset,
            x_extent, y_extent, message_length, key, key_length, key_lut,
            luminance_lut, detail_lut, thumbnail, original_data, referenceBitArray,
            metric, range, message, number_channels, bumps);
    }

    return(status);
}

//////////////////////////////////////
// read_8bit_single_channel_OLD_plus_color()
//////////////////////////////////////
void read_8bit_single_channel_OLD_plus_color(
    unsigned char *data, // input data to be recognized */
    long original_xdim, // it's x dimension */
    long original_ydim, // it's y dimension */
    long x_offset, // x offset of segment */
    long y_offset, // y offset of segment */
    long x_extent, // x extent of segment */
    long y_extent, // y extent of segment */
    int message_length, // length of message in BITS, also length of message
    string *key, // original 8 bit random key */
    unsigned char *key, // key_length often equal to data_length but not always */
    long key_length, // key_length often equal to data_length but not always */
    char *key_lut, // look up table mapping key value */
    float *luminance_lut, // look up table mapping the signature level to
    float *detail_lut, // look up table mapping the signature level to
    luminance*/

    unsigned char *thumbnail, // if available, use pointer, otherwise NULL */
    unsigned char *original_data, // if available, use pointer, otherwise NULL */

    const unsigned char *referenceBitArray, // bit array ptr: either the known message or
    estimate, // we will compute a return a crude metric indicating
    confidence.

    float *metric,
    float *range,
    unsigned char *message,
    int number_channels,
    int bumps
) {
    unsigned char *pkey, *pdata;
    long i, line, bit;
    int temp, status=1;
    float *key_value = new float[x_extent];
    float *data_float = new float[x_extent];
    float *orig_float = new float[x_extent];
    float *bit_total = new float[message_length];
    //float *bit_mag = new float[message_length];
    float *pkey_value, *pdata_float;
    float filter_cf = (float)0.5; // kludge for now
    double maxdiff = 40.0; // kludge for now

    int key_xlength = 1+(original_xdim-1)/bumps;
    for(i=0; i<message_length; i++)
    {
        bit_total[i] = (float) 0.0;
        //bit_mag[i] = (float) 0.0;
    }

    pdata = data;
    for(line=y_offset; line<(y_offset+y_extent); line++)

```

```

{
    /* FIRST: If either the original image or a thumbnail of the original is available,
    then use either a simple or "advanced" dot product to remove it; "advanced" refers
    to the idea that you may wish to adjust the gamma or higher order stuff */
    float it(pdata, data_float, x_extent, number_channels);
    //derivative threshold(data_float, x_extent, number_channels, maxdiff, filter_cf);
    //remove mean(data_float, x_extent);

    /* load key values */
    int key_offset = (line/bumps)*key_xlength;
    pkey = &key[key_offset + x_offset/bumps];
    pkey_value = key_value;
    if(bumps>1){
        for(i=x_offset; i<(x_offset+x_extent); i++){
            *(&pkey_value++) = (float){ (int)key_lut[ (int)*pkey ] };
            if( !((i+1)%bumps) ) pkey++;
        }
    }
    else {
        for(i=x_offset; i<(x_offset+x_extent); i++){
            *(&pkey_value++) = (float){ (int)key_lut[ (int)*pkey ] };
        }
    }
    pdata+=(number_channels*x_extent);

    /* now step through processed patch and perform simple or "advanced" correlation detection,
    keeping the resultant detection values in the accumulators for each bit of the
    message_length
    bits */
    pdata_float = data_float;
    pkey_value = key_value;
    float running_average = (float) 0.0;
    float ftemp;
    for (i = 0; i < MOV_AV_KERNEL; i++)
    {
        running_average += *(pdata_float++);
    }
    float mov_av = (float)MOV_AV_KERNEL;
    running_average /= mov_av;
    pdata_float = data_float;
    temp = MOV_AV_KERNEL/2;
    int temp1 = temp+1;
    if(bumps>1){
        for (i = x_offset; i < (x_offset + x_extent); i++)
        {
            if (i <= (x_offset + temp) || i >= (x_offset + x_extent - temp) );
            else
            {
                ftemp = (*(pdata_float + temp) - *(pdata_float - temp1)) / mov_av;
                running_average += ftemp;
            }
            bit = ( key_offset + i/bumps ) * message_length;
            ftemp = *(pdata_float++) - running_average;
            //bit_mag[bit] -= (*pkey_value * *pkey_value);
            bit_total[bit] += (ftemp * (*pkey_value++));
        }
    }
    else {
        for (i = x_offset; i < (x_offset + x_extent); i++)
        {
            if (i <= (x_offset + temp) || i >= (x_offset + x_extent - temp) );
            else
            {
                ftemp = (*(pdata_float + temp) - *(pdata_float - temp1)) / (float) MOV_AV_KERNEL;
                running_average += ftemp;
            }
            bit = ( key_offset + i ) * message_length;
            //bit_mag[bit] += (*pkey_value * *pkey_value);
            bit_total[bit] += ( (*pdata_float++) - running_average ) * (*pkey_value++);
        }
    }
    // time optimized version of above earlier code
    int key_foo = key_offset + x_offset;
    for(i=x_offset; i<(x_offset+temp); i++){
        bit = key_foo++ * message_length;
        bit_total[bit] += ( (*pdata_float++) - running_average ) * (*pkey_value++);
    }
    int temp2 = x_offset + x_extent - temp;
    float *pdata_float2 = data_float;
    float *pdata_float1 = spdata_float[temp];
    for(i=(x_offset+temp+1); i<temp2; i++){
        running_average += ( (*pdata_float1++) - *(pdata_float2++) )/mov_av;
        bit = key_foo++ * message_length;
        bit_total[bit] += ( (*pdata_float++) - running_average ) * (*pkey_value++);
    }
    for(i=0; i<temp; i++){
        bit = key_foo++ * message_length;
        bit_total[bit] += ( (*pdata_float++) - running_average ) * (*pkey_value++);
    }
}

```

```

    }
}
/* fill the message string based on bit_totals */
for(i=0; i<message_length; i++)
{
    if(bit_total[i]>0.0)
    {
        message[i]=1;
    }
    else
    {
        message[i]=0;
    }
}
/*
for (i = 0; i < message_length; i++)
{
    // Before normalizing by the magnitudes, be sure we aren't
    // dividing by zero (this happens for an image w/ zero energy.
    if (bit_mag[i] == (float)0.0)
        bit_mag[i] = epsilon;

    bit_total[i] /= (float) sqrt( (double) bit_mag[i] );
}
*/
// Compute the "crude metric", an estimate of rms spread of the
// bit level detector's results. The ReferenceBitArray is either
// the known message (if it was available to caller) or the
// newly computed estimate of the message.
*metric = get_crude_metric(referenceBitArray, bit_total, range, message_length);

delete [] data_float;
delete [] orig_float;
delete [] bit_total;
delete [] key_value;
//delete [] bit_mag;

return;

}

////////////////////
// float_it()
////////////////////
void float_it(unsigned char *data, float *data_float,
               long x_extent, int number_channels)
{
    unsigned char *pdata;
    long i;
    float *pdata;

    pdata = data;
    ppdata=data_float;
    if(number_channels == 1){
        for (i = 0; i < x_extent; i++)
            *(&pdata++) = (float) *(&data++);
    }
    else if (number_channels == 3) {
        for (i = 0; i < x_extent; i++){
            *pdata = (float) *(&data++);
            *pdata += (float) *(&data++);
            *(&pdata++) += (float) *(&data++);
        }
    }
}

////////////////////
// remove_mean()
////////////////////
void remove_mean(float *array, long length)
{
    long i;
    float total = (float) 0.0;
    for (i = 0; i < length; i++)
        total += array[i];
    total /= (float) length;
    for (i = 0; i < length; i++)
        array[i] -= total;
}

```



```

total /= ((float)y_extent * (float)x_extent);
for(i=0;i<y_extent;i++){
    pimage = &image[i*fftdim];
    for(j=0;j<x_extent;j++){
        *(pimage++)=total;
    }
}

float *pdetail_vector;
float *detail_vector = new float[x_extent];
int start = 5;
int stop = 500;
float scale = (float)0.5;
for(i=0;i<y_extent;i++){
    get_read_detail_vector(detail_vector,data,x_extent,i,y_extent,number_channels,start,stop,scale,image,fftdim);
    pdetail_vector = detail_vector;
    pimage = &image[i*fftdim];
    for(j=0;j<x_extent;j++){*(pimage++) += *(pdetail_vector++);}
    delete [] detail_vector;
}

//float filter_cf = (float)0.5; // kludge for now
//double maxdiff = 40.0; // kludge for now
//for(line=0; line<y_extent; line++){
//    // derivative_threshold(&image[line*fftdim], x_extent,1,maxdiff,filter_cf);
//}

// easy does the window ??
// for now, multiply the last four values near the edges by a linear ramp to zero, simply to avoid
total edge weirdnesses
int window_it=0;
if(window_it){
    if(x_extent > 10 && y_extent > 10){
        float mult[4]; *pmult;
        mult[0]=(float)0.2;mult[1]=(float)0.4;mult[2]=(float)0.6;mult[3]=(float)0.8;
        pmult = mult;
        for(i=1;i<5;i++){
            pimage = &image[(i-1)*fftdim];
            for(j=0;j<x_extent;j++){*(pimage++) *= *pmult;
            pmult++;
        }
        pmult = mult;
        for(i=1;i<5;i++){
            pimage = &image[(y_extent - i)*fftdim];
            for(j=0;j<x_extent;j++){*(pimage++) *= *pmult;
            pmult++;
        }
        for(i=0;i<y_extent;i++){
            pimage = &image[i*fftdim];
            pmult = mult;
            for(j=1;j<5;j++){*(pimage++) *= *pmult++};
            pimage = &image[(i+1)*fftdim-(x_extent-1)];
            pmult = mult;
            for(j=1;j<5;j++){*(pimage++) *= *pmult++};
        }
    }
}

// fft arrays
realfft2d_in_place(image,bits,0,wr,wl);

// filter them
// phase difference only to start
float magi,*preali,*pimaginary1;
// double power = 0.8;
preali=&image;pimaginary1=&image[fftdim];
for(i=0;i<(1+fftdim/2);i++){
    magi = (float)fabs( (double)*preali + (float)fabs( (double)*pimaginary1 ) );
    if(magi == (float)0.0){
        *(preali++)=(float)0.0;
        *(pimaginary1++) = (float)0.0;
    }
    else {
        //magi = (float)pow((double)magi,power);
        *(preali++) /= magi;
        *(pimaginary1++) /= magi;
    }
}
preali+=fftdim;
pimaginary1+=fftdim;
}

// remove low and/or high frequencies
// the DC should reside at row one, fftdim/2
int mo0 = 0;
if(mo0){
    int low = 1;
    int xcount=low*2-1;
    pimage = &image[(fftdim/2) - low +1];
    for(i=0;i<2*low;i++){
        for(j=0;j<xcount;j++){*(pimage++) = (float)0.0;
        pimage += (fftdim - xcount);
    }
}

// inverse fft
realfft2d_in_place(image,bits,1,wr,wl);
for(line=y_offset; line<(y_offset+y_extent); line++){
    // load key values */
    pkey = &key[(line/bumps) * key_xlength + x_offset/bumps];
    for(i=x_offset;i<(x_offset+x_extent);i++){
        *key_value[i-x_offset] = (float)( (int)key_lut[ (int)*pkey ] );
        if( (i+1)%bumps )pkey++;
    }
}

// now step through processed patch and perform simple or "advanced" correlation
detection, keeping the resultant detection values in the accumulators for each bit of the
message_length
bits */
pimage = &image[(line-y_offset)*fftdim];
pkey_value = key_value;
for(i=x_offset;i<(x_offset+x_extent);i++){
    {
        bit = ( (line/bumps)*key_xlength + i/bumps) % message_length;
        bit_mag[bit] += (*pkey_value * *pkey_value);
        bit_total[bit] += ( *pimage++) * (*pkey_value++);
    }
}

// * fill the message string based on bit_totals */
for(i=0; i<message_length; i++)
{
    if(bit_total[i]>0.0)
    {
        message[i]=1;
    }
    else
    {
        message[i]=0;
    }
}

for (i = 0; i < message_length; i++)
{
    // Before normalizing by the magnitudes, be sure we aren't
    // dividing by zero (this happens for an image w/ zero energy.
    if (bit_mag[i] == (float)0.0)
        bit_mag[i] = epsilon;
    bit_total[i] /= (float) sqrt( (double) bit_mag[i] );
}

// Compute the "crude metric", an estimate of rms spread of the
// bit level detector's results. The referenceBitArray is either
// the known message (if it was available to caller) or the
// newly computed estimate of the message.
*metric = get_crude_metric(referenceBitArray, bit_total,message_length);

delete [] bit_total;
delete [] bit_mag;
delete [] key_value;
delete [] image;
delete [] wr;
delete [] wi;
return;
}

////////////////////////////////////
// get_read_detail_vector()
////////////////////////////////////
// int get_read_detail_vector(
// float *detail_vector,

```

```

unsigned char *data,
int xdim,
int row,
int total_rows,
int number_channels,
int start,
int stop,
float scale,
float *image,
int fftdim
){
    unsigned char *p1;
    float *pdata,*p2;
    int i;
    float base,temp;
    float *pdetail_vector=detail_vector;

    // this function creates a "scaling" vector for the current scan line,
    // based on a crude metric of "local detail"
    if(number_channels == 1){
    }
    else if(number_channels == 3)
    {
        pdata = image[row*fftdim];
        if(row == 0)p1 = &data[3*row*xdim];
        else p1 = &data[3*(row-1)*xdim];
        if(row == (total_rows-1))p2 = &image[row*fftdim];
        else p2 = &image[(row+1)*fftdim];
        // perform first and last elements outside loop so that an internal if statement is avoided
        base = (float)*(p1++) ,base+=(float)*(p1++) ,base+=(float)*(p1++);
        base+=*(p2++);
        base+= (float)2.0 * *(pdata+1);
        temp = base/(float)4.0 - *(pdata++);
        float denom = (float)(stop-start)/((float)1.0-scale);
        float mult;
        base = (float)fabs( (double)temp );
        if( base > (float)start ){
            if(base > (float)stop)mult = (float)1.0 - scale;
            else mult = (base - (float)start)/denom;
            * (pdetail_vector++) = mult * temp;
        }
        else * (pdetail_vector++) = (float)0.0;
        for(i=1;i<(xdim-1);i++)
        {
            base = (float)*(p1++) ,base+=(float)*(p1++) ,base+=(float)*(p1++);
            base+=*(p2++);
            base+=*(pdata+1);
            base+=*(p2+1);
            temp = base/(float)4.0 - *(pdata++);
            base = (float)fabs( (double)temp );
            if( base > (float)start ){
                if(base > (float)stop)mult = (float)1.0 - scale;
                else mult = (base - (float)start)/denom;
                * (pdetail_vector++) = mult * temp;
            }
            else * (pdetail_vector++) = (float)0.0;
        }
        base = (float)*(p1++) ,base+=(float)*(p1++) ,base+=(float)*p1;
        base+=*(p2);
        base+= (float)2.0 * *(pdata-1);
        temp = base/(float)4.0 - *pdata;
        base = (float)fabs( (double)temp );
        if( base > (float)start ){
            if(base > (float)stop)mult = (float)1.0 - scale;
            else mult = (base - (float)start)/denom;
            *pdetail_vector = mult * temp;
        }
        else *pdetail_vector = (float)0.0;
    }
    return(1);
}

/////////////////////////////////////////////////
// Header file for the Reader core algorithm functions.
/////////////////////////////////////////////////
#ifndef READ_H
#define READ_H

#define SECOND_THRESHOLD (float) 20.0
#define FIRST_THRESHOLD (float) 20.0

#define MOV_AV_KERNEL 5

int derivative_threshold(float *data, long length, int number_channels,double maxdiff,
float filter_cf);

```

```

//void float_it(unsigned char *data, float *data_float, long x_extent);
void float_it(unsigned char *data, float *data_float,
              long x_extent, int number_channels);
void remove_mean(float *array, long length);
float get_crude_metric(const unsigned char *actual_message,
                      float *bit_total,
                      float *range,
                      int message_length);

int read_8bit_single_channel_or_color(
    unsigned char *data,          /* input data to be recognized */
    long original_xdim,          /* it's x dimension */
    long original_ydim,          /* it's y dimension */
    long x_offset,               /* x offset of segment */
    long y_offset,               /* y offset of segment */
    long x_extent,               /* x extent of segment */
    long y_extent,               /* y extent of segment */
    long message_length,         /* length of message in BITS, also length of message
    string */
    unsigned char *key,          /* original 8 bit random key */
    long key_length,             /* key_length often equal to data_length but not always */
    /* unused */
    char *key_lut,               /* look up table mapping key value */
    float *luminance_lut,        /* look up table mapping the signature level to
    luminance */
    float *detail_lut,           /* look up table mapping the signature level to detail */
    unsigned char *thumbnail,    /* if available, use pointer, otherwise NULL */
    unsigned char *original_data, /* if available, use pointer, otherwise NULL */
    const unsigned char *referenceBitArray, /* bit array ptr: either the known message or
    estimate. */
    float *metric,               /* we will compute a return a crude metric indicating
    confidence. */
    float *range,                /* output: either 0 or 1, i.e. inefficient but simple */
    int number_channels,          /* generally for B&W=1 vs. color == 3
    int reading_mode,
    int bumps);

```

```

void read_8bit_single_channel_OLD_plus_color(
    unsigned char *data,          /* input data to be recognized */
    long original_xdim,          /* it's x dimension */
    long original_ydim,          /* it's y dimension */
    long x_offset,               /* x offset of segment */
    long y_offset,               /* y offset of segment */
    long x_extent,               /* x extent of segment */
    long y_extent,               /* y extent of segment */
    int message_length,          /* length of message in BITS, also length of message
    string,                       */
    unsigned char *key,          /* original 8 bit random key */
    long key_length,             /* key_length often equal to data_length but not always */
    /**unused**key_lut,          /* look up table mapping key value */
    float *luminance_lut,        /* look up table mapping the signature level to
    luminance/,                  /* look up table mapping the signature level to
    float *detail_lut,
    luminance/,

    unsigned char *thumbnail,    /* if available, use pointer, otherwise NULL */
    unsigned char *original_data, /* if available, use pointer, otherwise NULL */

    const unsigned char *referenceBitarray, /* bit array ptr: either the known message or
    estimate,                       */
    float *metric,                 /* we will compute a return a crude metric indicating
    confidence,

    float *range,
    unsigned char *message,
    int number_channels,
    int bytes_per_channel)

```

```

void read_super(
    unsigned char *data,
    long original_xdim,
    long original_ydim,
    long x_offset,
    long y_offset,
    long x_extent,
    long y_extent,
    long message_length,
    string *,
    unsigned char *key,
    long key_length,
    /**unused**/,
    char *key_lut,
    float *luminance_lut,
    luminance,
    /** input data to be recognized */
    /** it's x dimension */
    /** it's y dimension */
    /** x offset of segment */
    /** y offset of segment */
    /** x extent of segment */
    /** y extent of segment */
    /** length of message in BITS, also length of message */
    /** original 8 bit random key */
    /** key_length often equal to data_length but not always */
    /** look up table mapping key value */
    /** look up table mapping the signature level to

```

```

float *detail_lut,
/* look up table mapping the signature level to luminance*/

unsigned char *thumbnail,
/* if available, use pointer, otherwise NULL*/
unsigned char *original_data,
/* if available, use pointer, otherwise NULL*/

const unsigned char *referenceBitArray, // bit array ptr: either the known message or estimate.
float *metric,
float *range,
unsigned char *message,
int number_channels,
int bumps);

/* output: either 0 or 1, i.e. inefficient but simple */

```

```

int get_read_detail_vector(
float *detail_vector,
unsigned char *data,
int xdim,
int row,
int total_rows,
int number_channels,
int start,
int stop,
float scale,
float *image,
int fftdim
);

```

```

#endif // READ_H

```

READDLG.CPP

```

// readdlg.cpp : implementation file
//

```

```

#include "stdafx.h"
#include "signer.h"
#include "readdlg.h"

```

```

#ifdef _DEBUG
#define THIS_FILE
static char _BASED_CODE THIS_FILE[] = __FILE__;
#endif

```

```

// ReadDlg dialog
//

```

```

// ReadDlg()
//

```

```

// Constructor for the Reader Parameters Dialog object. A ReadDlg
// object is created to manage a dialog in which the user is able
// to set the parameters used by the Reader and associated core
// algorithms.
//

```

```

ReadDlg::ReadDlg(CWnd* pParent /*=NULL*/)
{
    : CDialog(ReadDlg::IDD, pParent)

```

```

{
    //{{AFX_DATA_INIT(ReadDlg)
    m_user_key = 0;
    m_msg_length = 0;
    m_gain = (float) 0.0;
    m_bump_size = 0;
    m_detail_lut_scale = 0.0f;
    //}}AFX_DATA_INIT
}

```

```

void ReadDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);

```

```

//{{AFX_DATA_MAP(ReadDlg)
DDX_Text(pDX, IDC_READ_KEY, m_user_key);
DDV_MinMaxInt(pDX, m_user_key, 0, 65535);
DDX_Text(pDX, IDC_READ_LENGTH, m_msg_length);
DDV_MinMaxInt(pDX, m_msg_length, 1, 65535);
DDX_Text(pDX, IDC_READ_GAIN, m_gain);
DDV_MinMaxFloat(pDX, m_gain, 1.e-003f, 1.e+006f);
DDX_Text(pDX, IDC_READ_SIZE, m_bump_size);
DDV_MinMaxInt(pDX, m_bump_size, 1, 256);
DDX_Text(pDX, IDC_DETAIL_LUT_SCALE, m_detail_lut_scale);
DDV_MinMaxFloat(pDX, m_detail_lut_scale, 1.e-003f, 1.e+006f);
//}}AFX_DATA_MAP

```

```

BEGIN_MESSAGE_MAP(ReadDlg, CDialog)
    //{{AFX_MSG_MAP(ReadDlg)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

```

```

// ReadDlg message handlers
//
void ReadDlg::OnOK()
{
    CDialog::OnOK();
}

```

READDLG.H

```

// readdlg.h : header file
//

```

```

// ReadDlg dialog
//

```

```

class ReadDlg : public CDialog
{
// Construction
public:
    ReadDlg(CWnd* pParent = NULL); // standard constructor

```

```

// Dialog Data
//{{AFX_DATA(ReadDlg)
enum { IDD = IDD_READ_DIALOG };
UINT m_user_key;
UINT m_msg_length;
float m_gain;
int m_bump_size;
float m_detail_lut_scale;
//}}AFX_DATA

```

```

// Implementation
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support

```

```

// Generated message map functions
//{{AFX_MSG(ReadDlg)
virtual void OnOK();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

```

RESOURCE.H

```

//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
//
// Used by Signer.rc

```

```

#define IDR_MAINFRAME 2
#define IDR_DIBTYPE 3
#define IDD_ABOUTBOX 100
#define IDC_MESSAGE 101
#define IDD_PARAMS_DIALOG 101
#define IDC_GAIN 102
#define IDC_GAIN_LABEL 103
#define IDC_READ_DIALOG 103
#define IDC_MESSAGE_LABEL 104
#define IDC_EDIT_GAIN 106
#define IDC_EDIT_GAMMA 107
#define IDC_EDIT_KEY 108
#define IDC_READ_KEY 110
#define IDC_READ_LENGTH 111
#define IDC_READ_GAIN 112
#define IDC_READ_SIZE 115
#define IDC_READ_SCALE 118
#define IDC_DETAIL_LUT_SCALE 121
#define IDC_EDIT_SETTINGS 32769
#define ID_VIEW_SETTINGS 32770
#define ID_VIEW_UNSIGNED 32771
#define ID_VIEW_SNOW 32772
#define ID_VIEW_SNOWY_IMAGE 32773
#define ID_VIEW_STATUS 32774
#define ID_SETTINGS_SIGNER 32775
#define ID_SETTINGS_READER 32776
#define ID_SETTINGS_REGISTRY 32777
#define ID_SETTINGS_AUTOPRINTREPORT 32778
#define ID_SETTINGS_AUTOPRINT 32779
#define ID_OPTIONS_AUTOREAD 32780
#define ID_SETTINGS_AUTOREAD 32781
#define ID_CONTROLS_ALIGN 32782
#define ID_SETTINGS_ALIGN 32783

```

```

// Next default values for new objects
//

```

```

#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE 106
#define _APS_NEXT_COMMAND_VALUE 32784
#define _APS_NEXT_CONTROL_VALUE 122
#define _APS_NEXT_SYMED_VALUE 102
#endif
#endif

// FILE: Sign.cpp
// DESCRIPTION:
// Core signing functions of the digimarc technology.
// Created July 1995.
// Copyright (C) 1996 Digimarc Corporation, all rights reserved.
//
// This function loads the scaling factor based on luminance */
int load_luminance_lut( float *luminance_lut, float gamma) // explicitly written for 8 bit
{
    int i, status=1;
    for(i=1; i<256; i++)
    {
        luminance_lut[i] = (float) 0.; /* don't put any signature energy into zero luminance (black) */
        return(status);
    }
}

// load_key_lut()
// This function just assigns mainly 0's, 1's, -1's, 2's and -2's
// to the key values, scaled by the scale_point
// scale_point is a simple integer between 1 and 127
// about 30 to 50 should be about right for first tests
float load_key_lut(char *key_lut, float gain)
{
    int i, base_gain, ifraction;
    float rms, fraction;
    gain /= (float)100.0;
    rms = gain;
    base_gain = (int)gain;
    fraction = gain;
    ifraction = (int)((float)base_gain);
    if(ifraction == 0)
    {
        for(i=0; i<128; i++) key_lut[i] = (char)base_gain;
        for(i=0; i<128; i++) key_lut[i+128] = -(char)base_gain;
    }
    else
    {
        for(i=0; i<(128-ifraction); i++)
        {
            key_lut[i] = (char)base_gain;
            key_lut[i+128] = -(char)base_gain;
        }
        for(i=(128-ifraction); i<128; i++)
        {
            key_lut[i] = (char)(base_gain+1);
            key_lut[i+128] = -(char)(base_gain+1);
        }
    }
    return( rms );
}

// The following functions are core algorithms which include
// 1) additional capabilities for signing Color images, and
// 2)

```

```

// get_detail_vector()
//
// get_detail_vector(
// float *detail_vector,
// unsigned char *data,
// int xdim,
// int row,
// int total_rows,
// float *detail_lut,
// int number_channels
// )
{
    unsigned char *pdata, *pl, *p2;
    int base, temp, i;
    float *pdetail_vector=detail_vector;
    // this function creates a "scaling" vector for the current scan line,
    // based on a crude metric of "local detail"
    if(number_channels == 1){
        pdata = data;
        if(row == 0) p1 = data;
        else p1 = data - xdim;
        if(row == (total_rows-1)) p2 = data;
        else p2 = data + xdim;
        // perform first and last elements outside loop so that an internal if statement is
        avoided
        base = (int)*(pdata++);
        temp = abs(base- (int)*(p1++));
        temp += abs(base- (int)*(p2++));
        temp += 2*abs(base- (int)*pdata);
        *pdetail_vector++ = detail_lut[temp]; // make sure it goes up to 1024 elements
        for(i=1; i<(xdim-1); i++){
            base = (int)*(pdata++);
            temp = abs(base- (int)*(p1++));
            temp += abs(base- (int)*(p2++));
            temp += abs(base- (int)*pdata);
            temp += abs(base- (int)*(pdata-2));
            *pdetail_vector++ = detail_lut[temp];
        }
        base = (int)*pdata;
        temp = abs(base- (int)*p1);
        temp += abs(base- (int)*p2);
        temp += 2*abs(base- (int)*(pdata-1));
        *pdetail_vector = detail_lut[temp]; // make sure it goes up to 1024 elements
    }
    else if(number_channels == 3)
    {
        // use the green channel only just for speed's sake
        pdata = data+1;
        if(row == 0) p1 = data+1;
        else p1 = data+1 - 3*xdim;
        if(row == (total_rows-1)) p2 = data+1;
        else p2 = data+1 + 3*xdim;
        // Perform first and last elements outside loop so that an internal if statement is
        avoided
        base = (int)*pdata;
        temp = abs(base- (int)*p1);
        temp += abs(base- (int)*p2);
        temp += 2*abs(base- (int)*pdata);
        *pdetail_vector++ = detail_lut[temp]; // make sure it goes up to 1024 elements
        for(i=1; i<(xdim-1); i++){
            base = (int)*pdata;
            temp = abs(base- (int)*p1);
            temp += abs(base- (int)*p2);
            temp += abs(base- (int)*pdata);
            temp += abs(base- (int)*(pdata-6));
            *pdetail_vector++ = detail_lut[temp];
        }
        base = (int)*pdata;
        temp = abs(base- (int)*p1);
        temp += abs(base- (int)*p2);
        temp += 2*abs(base- (int)*(pdata-3));
        *pdetail_vector = detail_lut[temp]; // make sure it goes up to 1024 elements
    }
    return(1);
}

// load_detail_lut()
// This function loads the scaling factor based on local detail
// int load_detail_lut( float *detail_lut, float scale) // explicitly written for 8 bit
{
    int i, status=1;
    float length=(float) (DETAIL_STOP-DETAIL_START);

```

```

scale /= (float)100.0;
scale*=DETAIL_NORMALIZER;
for(i=0;i<DETAIL_START;i++)detail_lut[i]=(float)1.0;
for(i=DETAIL_START; i<DETAIL_STOP; i++)
{
    detail_lut[i] = (float)1.0 + scale*((float)(i-DETAIL_START)/length);
}
for(i=DETAIL_STOP;i<DETAIL_TOTAL;i++)detail_lut[i]=detail_lut[DETAIL_STOP-1];

return(status);
}

// sign_8bit_single_channel_or_color()
// written for the march 1996 bump incarnation
int sign_8bit_single_channel_or_color(
    unsigned char *data,
    long data_length,
    long ydim,
    long xdim,
    unsigned char *message,
    int message_length,
    unsigned char *key,
    long key_length,
    char *key_lut,
    float *luminance_lut,
    float *detail_lut,
    int signing_mode,
    unsigned char *data_out,
    int number_channels,
    images
)
{
    int bumps; // added in March 1996 to implement bumps
    unsigned char *pdata;
    unsigned char *p_out;
    unsigned char *pkey;
    unsigned char *pmessage;
    long i;
    int j,k;
    int lum_change;
    float ftemp,delta;
    float *detail_vector = new float(xdim);
    float *pdetail_vector,local_gain;
    int key_xlength;

    key_xlength = 1+(xdim-1)/bumps;
    if(number_channels == 1){
        pdata = data;
        p_out = data_out;
        for(i=0;i<ydim;i++){
            // load local detail values for this row
            get_detail_vector(detail_vector,pdata,xdim,i,ydim,detail_lut,number_channels);
            pdetail_vector = detail_vector;
            pkey=key[(i/bumps)*key_xlength];
            pmessage = &message[(i/bumps)*key_xlength*message_length];
            for(j=0;j<xdim;j++){
                if(lum_change == 0){
                    lum_change = key_lut[(int)*pkey];
                    memcpy(p_out,pdata,3*sizeof(unsigned char));
                    pdata+=3;
                    p_out+=3;
                    pdetail_vector++;
                } else {
                    local_gain = *(pdetail_vector++) * luminance_lut[(int)*pdata+1];
                    if( abs(lum_change) > 1 ){ // this is the anti-sparklies check
                        if( local_gain > (float)3.5 ){
                            if(lum_change > 0)lum_change = 1;
                            else lum_change = -1;
                        }
                    }
                }
                *p_out++ = *(pdata++);
                pdetail_vector++;
            }
        }
        return(status);
    }
}

// sign_8bit_single_channel_or_color()
// written for the march 1996 bump incarnation
int sign_8bit_single_channel_or_color(
    unsigned char *data,
    long data_length,
    long ydim,
    long xdim,
    unsigned char *message,
    int message_length,
    unsigned char *key,
    long key_length,
    char *key_lut,
    float *luminance_lut,
    float *detail_lut,
    int signing_mode,
    unsigned char *data_out,
    int number_channels,
    images
)
{
    int bumps; // added in March 1996 to implement bumps
    unsigned char *pdata;
    unsigned char *p_out;
    unsigned char *pkey;
    unsigned char *pmessage;
    long i;
    int j,k;
    int lum_change;
    float ftemp,delta;
    float *detail_vector = new float(xdim);
    float *pdetail_vector,local_gain;
    int key_xlength;

    key_xlength = 1+(xdim-1)/bumps;
    if(number_channels == 1){
        pdata = data;
        p_out = data_out;
        for(i=0;i<ydim;i++){
            // load local detail values for this row
            get_detail_vector(detail_vector,pdata,xdim,i,ydim,detail_lut,number_channels);
            pdetail_vector = detail_vector;
            pkey=key[(i/bumps)*key_xlength];
            pmessage = &message[(i/bumps)*key_xlength*message_length];
            for(j=0;j<xdim;j++){
                if(lum_change == 0){
                    lum_change = key_lut[(int)*pkey];
                    memcpy(p_out,pdata,3*sizeof(unsigned char));
                    pdata+=3;
                    p_out+=3;
                    pdetail_vector++;
                } else {
                    local_gain = *(pdetail_vector++) * luminance_lut[(int)*pdata+1];
                    if( abs(lum_change) > 1 ){ // this is the anti-sparklies check
                        if( local_gain > (float)3.5 ){
                            if(lum_change > 0)lum_change = 1;
                            else lum_change = -1;
                        }
                    }
                }
                *p_out++ = *(pdata++);
                pdetail_vector++;
            }
        }
        return(status);
    }
}

// sign_8bit_single_channel_or_color()
// written for the march 1996 bump incarnation
int sign_8bit_single_channel_or_color(
    unsigned char *data,
    long data_length,
    long ydim,
    long xdim,
    unsigned char *message,
    int message_length,
    unsigned char *key,
    long key_length,
    char *key_lut,
    float *luminance_lut,
    float *detail_lut,
    int signing_mode,
    unsigned char *data_out,
    int number_channels,
    images
)
{
    int bumps; // added in March 1996 to implement bumps
    unsigned char *pdata;
    unsigned char *p_out;
    unsigned char *pkey;
    unsigned char *pmessage;
    long i;
    int j,k;
    int lum_change;
    float ftemp,delta;
    float *detail_vector = new float(xdim);
    float *pdetail_vector,local_gain;
    int key_xlength;

    key_xlength = 1+(xdim-1)/bumps;
    if(number_channels == 1){
        pdata = data;
        p_out = data_out;
        for(i=0;i<ydim;i++){
            // load local detail values for this row
            get_detail_vector(detail_vector,pdata,xdim,i,ydim,detail_lut,number_channels);
            pdetail_vector = detail_vector;
            pkey=key[(i/bumps)*key_xlength];
            pmessage = &message[(i/bumps)*key_xlength*message_length];
            for(j=0;j<xdim;j++){
                if(lum_change == 0){
                    lum_change = key_lut[(int)*pkey];
                    memcpy(p_out,pdata,3*sizeof(unsigned char));
                    pdata+=3;
                    p_out+=3;
                    pdetail_vector++;
                } else {
                    local_gain = *(pdetail_vector++) * luminance_lut[(int)*pdata+1];
                    if( abs(lum_change) > 1 ){ // this is the anti-sparklies check
                        if( local_gain > (float)3.5 ){
                            if(lum_change > 0)lum_change = 1;
                            else lum_change = -1;
                        }
                    }
                }
                *p_out++ = *(pdata++);
                pdetail_vector++;
            }
        }
        return(status);
    }
}

```

```

pkey++;
if( (((i/bumps)*key_xlength+j/bumps)*message_length) == (message_length-1) )
/* time to restart message */
{
    pmessage = message;
    else pmessage++;
}
}
}
else if(number_channels == 3){
    // data_length is assumed to be the number of pixels, not the number of data bytes
    // RGB packing is assumed, in that order, 3 bytes in a row per pixel: R G B
    if(signing_mode == STANDARD){
        pdata = data;
        p_out = data_out;
        for(i=0;i<ydim;i++){
            // load local detail values for this row
            get_detail_vector(detail_vector,pdata,xdim,i,ydim,detail_lut,number_channels);
            pdetail_vector = detail_vector;
            pkey=key[(i/bumps)*key_xlength];
            pmessage = &message[(i/bumps)*key_xlength*message_length];
            for(j=0;j<xdim;j++){
                lum_change = key_lut[(int)*pkey];
                if(lum_change == 0){
                    memcpy(p_out,pdata,3*sizeof(unsigned char));
                    pdata+=3;
                    p_out+=3;
                    pdetail_vector++;
                } else {
                    local_gain = *(pdetail_vector++) * luminance_lut[(int)*pdata+1];
                    if( abs(lum_change) > 1 ){ // this is the anti-sparklies check
                        if( local_gain > (float)3.5 ){
                            if(lum_change > 0)lum_change = 1;
                            else lum_change = -1;
                        }
                    }
                }
                *p_out++ = *(pdata++);
                pdetail_vector++;
            }
        }
        delta = (float)lum_change * local_gain;
        if( !(*pmessage) )
            delta = -delta; /* invert current snowy image luminance value ...
key */
        for(k=0;k<3;k++){
            ftemp = (float)*(pdata++) + delta;
            if(ftemp > (float)255.0)*(p_out++) = (unsigned char)255;
            else if(ftemp < (float)0.0)*(p_out++) = (unsigned char)0;
            else *(p_out++) = (unsigned char)(ftemp*(float)0.5);
        }
    }
    if( (((j+1)%bumps) == 0 ) {
        pkey++;
        if( (((i/bumps)*key_xlength+j/bumps)*message_length) ==
/* time to restart message */
        {
            pmessage = message;
            else pmessage++;
        }
    }
    return(status);
}

// FILE: Sign.h
// DESCRIPTION:
// Header file for the signing core algorithms. Callers of the signing
// functions should include this file.
// Copyright (C) 1996 Digimarc Corporation, all rights reserved.
// #ifndef SIGN_H
// #define SIGN_H
// These are the possible settings of the "signing_mode" argument
// #define STANDARD 0

```

```

#define STRICT_LUMINANCE 1
#define LUMINANCE_RED (float)0.31
#define LUMINANCE_GREEN (float)0.59
#define LUMINANCE_BLUE (float)0.11
#define DETAIL_START 20
#define DETAIL_STOP 200
#define DETAIL_TOTAL 1024
#define DETAIL_NORMALIZER (float)7.0

int load_luminance_lut( float *luminance_lut, float gamma );

float load_key_lut( char *key_lut , float gain);

// The following function prototypes correspond to the more
// advanced signing algorithms and color image signing capabilities
// added in February 1996.

int get_detail_vector(float *detail_vector,
                     unsigned char *data,
                     int xdim,
                     int row,
                     int total_rows,
                     float *luminance_lut,
                     int number_channels);

int load_detail_lut( float *detail_lut, float scale); // explicitly written for 8 bit

int sign_8bit_single_channel_or_color(
    unsigned char *data, // input data to be signed
    long data_length, // it's length
    long xdim, // it's x dimension
    long ydim, // it's y dimension
    unsigned char *message, // either 0 or 1, i.e. inefficient but simple
    int message_length, // length of message in BITS, also length of message string
    unsigned char *key, // 8 bit random key, uniformly distributed
    long key_length, // key length offset, equal to data length but not always *unused*
    char *key_lut, // look up table mapping key value
    float *luminance_lut, // look up table mapping the scaling to luminance values
    int *detail_lut, // look up table mapping the scaling to luminance values
    int signing_mode, // signed output data in same length and format as input
    unsigned char *data_out, // added in late february 1996 to begin work on 3 color 24 bit color
    int number_channels, // added in March 1996
    int bumps
);

#endif // SIGN_H

// FILE: Signdoc.cpp
// DESCRIPTION:
// Implementation file for the Document class of the DigiMARC Signer.
// This defines the implementation of the document class
// for the Signer. Under the Microsoft Foundation Class (MFC) architecture,
// the Document/View model is the preferred method. This header file
// defines our additions to the Generic Document class created by the
// Visual C++ wizards.
// Copyright (C) 1996 DigiMARC Corporation, all rights reserved.

#include "stdafx.h"
#include "signer.h"
#include "limits.h"

#include "signdoc.h"
#include "signview.h"

#include "cokey.h"
#include "image.h"
#include "sign.h"
#include "read.h"
#include "align.h"

#include "parmsdlg.h"
#include "readdlg.h"

// For the Signer Parameters dialog object
// For the Reader Parameters dialog object

#include "afxpriv.h"
#include "afxext.h"
#include "mainfrm.h"

```

```

#include <strstream.h>
#include <fstream.h>

#ifdef DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

// CDibDoc
// Message Map setup.
// CDibDoc
IMPLEMENT_DYNCREATE(CDibDoc, CDocument)

BEGIN_MESSAGE_MAP(CDibDoc, CDocument)
//{{AFX_MSG_MAP(CDibDoc)
ON_COMMAND(ID_SETTINGS_SIGNER, OnSettingsSigner)
ON_COMMAND(ID_SETTINGS_AUTOPRINT, OnSettingsAutoprint)
ON_UPDATE_COMMAND_UI(ID_SETTINGS_AUTOPRINT, OnUpdateSettingsAutoprint)
ON_COMMAND(ID_SETTINGS_READER, OnSettingsReader)
ON_UPDATE_COMMAND_UI(ID_SETTINGS_READER, OnUpdateSettingsAutoprint)
ON_COMMAND(ID_SETTINGS_AUTOREAD, OnSettingsAutoread)
ON_UPDATE_COMMAND_UI(ID_SETTINGS_AUTOREAD, OnUpdateSettingsAutoread)
ON_COMMAND(ID_SETTINGS_ALIGN, OnSettingsAlign)
ON_UPDATE_COMMAND_UI(ID_FILE_SAVE_AS, OnUpdateFileSaveAs)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

// CDibDoc::CDibDoc()
// Constructor for the Signer Document class
// CDibDoc::CDibDoc()
// m_hDIB = NULL; // dummy value to make CScrollView happy
// m_pDIB = NULL;
// m_sizeDoc = CSize(1,1);
// m_hOriginalDIB = NULL;
// m_hSnowyDIB = NULL;
// m_hSignedDIB = NULL;
// m_pImage = NULL;
// m_pAlignedImage = NULL;
// m_pAlign = NULL;
// m_pParams = NULL;
// m_pPackedMsg = NULL;

// Toggles controlled from the "options" menu
m_autoprint = FALSE;
m_autoread = (CDibLookApp *)AfxGetApp()->m_autoread;
m_state = NO_IMAGE;
m_filename = "\0";

// CDibDoc::~CDibDoc()
// Destructor for the Signer Document class.
// CDibDoc::~CDibDoc()
// if (m_hOriginalDIB != NULL)
// {
// ::GlobalFree((HGLOBAL) m_hOriginalDIB);
// m_hOriginalDIB = NULL;
// if (m_pDIB != NULL)
// {
// delete m_pDIB;
// }
// if (m_hSnowyDIB != NULL)
// {
// ::GlobalFree((HGLOBAL) m_hSnowyDIB);
// m_hSnowyDIB = NULL;
// if (m_hSignedDIB != NULL)
// {
// ::GlobalFree((HGLOBAL) m_hSignedDIB);
// m_hSignedDIB = NULL;
// if (m_pPackedMsg != NULL)
// delete m_pPackedMsg;
// if (m_pAlign != NULL)

```



```

// Set pointer to the DIB of the image which is to be saved.
if (view_type == ORIGINAL_VIEW)
    hSavedDIB = m_hOriginalDIB;
else if (view_type == SIGNED_VIEW)
    hSavedDIB = m_hSignedDIB;
else if (view_type == ALIGNED_VIEW)
    hSavedDIB = m_hAlignedImage->GetHDIIB();
else if (view_type == STATUS_VIEW)
{
    // This is the unusual case where we are not saving a DIB.
    // Instead, we write out the character strings of the status view.
    file.Close(); // close the binary file, create ostream instead
    ostream of(pszPathName); // Text output file stream
    CDibView* stat_view; // For in-memory formatting of the string
    stat_view = GetActiveView();
    stat_view->CreateStatusStream(stat_stream);
    // While the status information to the file
    of << stat_stream.str();
    of.close();
    delete stat_stream.str(); // Once we use .str, we have to delete it.
    return TRUE;
}

TRY
{
    BeginWaitCursor();
    bSuccess = ::SaveDIB(hSavedDIB, file);
    file.Close();
}
CATCH (CException, eSave)
{
    file.Abort(); // will not throw an exception
    EndWaitCursor();
    ReportSaveLoadException(pszPathName, eSave,
        TRUE, AFX_IDP_FAILED_TO_SAVE_DOC);
    return FALSE;
}
END_CATCH

EndWaitCursor();
SetModifiedFlag(FALSE); // back to unmodified
if (!bSuccess)
{
    // may be other-style DIB (load supported but not save)
    // or other problem in SaveDIB
    MessageBox(NULL, "Couldn't save DIB", NULL,
        MB_ICONINFORMATION | MB_OK);
}

if (m_state == IMAGE_SIGNED_AND_VERIFIED)
{
    m_state = IMAGE_SIGNED_AND_SAVED;
    // Save the name of the saved file.
    m_filename = pszPathName;
    // If the user switch is set, create a "Status view" (iff it doesn't
    // already exist), and print it.
    if (m_autoprint)
    {
        CDibView *p_status_view;
        p_status_view = (CDibView*) CreateUniqueView(STATUS_VIEW);
        p_status_view->OnFilePrint();
    }
    else
        UpdateAllViews(NULL); // If status view present, needs update
    return bSuccess;
}

void CDibDoc::ReplaceHDIIB(HDIIB hDIB)
{
    if (m_hOriginalDIB != NULL)
    {
        ::GlobalFree((HGLOBAL) m_hOriginalDIB);
        m_hOriginalDIB = hDIB;
    }
}

////////////////////////////////////
// CDbDoc diagnostics
#ifdef DEBUG
void CDibDoc::AssertValid() const
{
    CDocument::AssertValid();
}
#endif

```



```

// MakeSnow()
// Creates a snow image, and sets the member variable m_hSnowyDIB, which
// is a DIB handle to the new snow image DIB. The snow image which is
// created is sized based on the parent DIB handle passed in, and it
// has all the same bitmap header and palette stuff.
// GlobalUnlock(HGLOBAL) m_hSnowyDIB;
// First shot at a function which calls the signer core algorithms
void CDbDoc::MakeSnow(HDIB hParentDIB)
{
    int cxDIB, cyDIB;
    long num_pixels, num_colors;
    DWORD total_size, image_byte;
    LPSTR lpDIB, lpSnowyDIB;
    LPBITMAPINFOHEADER lpSnowyDIBHdr;
    HPSTR hPSnowyDIBBits;
    HPSTR src_data, dest_data;

    // Huge ptrs for copying the image.

    // HDIB hOriginalDIB = GetOriginalHDIB();
    if (hParentDIB == NULL)
        return;

    // Get the size of the parent DIB
    total_size = GlobalSize((HGLOBAL) hParentDIB);

    // Create space for the snow image (on 1st call only).
    if (m_hSnowyDIB == NULL)
    {
        m_hSnowyDIB = (HDIB)::GlobalAlloc(GMEM_MOVEABLE | GMEM_ZEROINIT, total_size);
        if (m_hSnowyDIB == 0)
        {
            MessageBox(NULL,
                "Insufficient memory is available for the \"snow image\"",
                "Digitarc Signer Warning",
                MB_ICONINFORMATION | MB_OK);
            return;
        }

        // Lock the two DIBs in memory
        lpDIB = (LPSTR)::GlobalLock((HGLOBAL) hParentDIB);
        lpSnowyDIB = (LPSTR)::GlobalLock((HGLOBAL) m_hSnowyDIB);

        src_data = (char *) lpDIB;
        dest_data = (char *) lpSnowyDIB;

        // Copy the BITMAPINFOHEADER, palette, and actual image byte data by byte.
        for (image_byte = 0; image_byte < total_size; image_byte++)
        {
            *dest_data++ = *src_data++;
        }

        // For debug: reset the pointers.
        src_data = (char *) lpDIB;
        dest_data = (char *) lpSnowyDIB;
        if (*src_data != *dest_data)
            TRACE("DEBUG: after copy into snow image, 1st chars aren't equal!\n");

        // We are now all done w/ the Parent DIB. Unlock it.
        ::GlobalUnlock((HGLOBAL) hParentDIB);

        // Get ptr to the snowy dib header space.
        lpSnowyDIBHdr = (LPBITMAPINFOHEADER) lpSnowyDIB;

        lpSnowyDIBBits = ::FindDIBBits(lpSnowyDIB);

        cxDIB = (int)::DIBWidth(lpSnowyDIB); // X size of DIB
        cyDIB = (int)::DIBHeight(lpSnowyDIB); // Y size of DIB

        num_pixels = (long) cxDIB * cyDIB;
        num_colors = ::DIBNumColors(lpSnowyDIB);

        if (lpSnowyDIBHdr->biCompression != 0)
        {
            TRACE("Can't cope with compressed image (compression = %d)\n",
                lpSnowyDIBHdr->biCompression);
            ::GlobalUnlock((HGLOBAL) m_hSnowyDIB);
            return;
        }

        TRACE("width = %d, height = %d, num_pixels = %d\n", cxDIB, cyDIB, num_pixels);
        TRACE("num_colors = %d\n", num_colors);

        if (m_BitsPerPixel != 8 && m_BitsPerPixel != 24)
    {
        TRACE("At this time, only build snow image for 8 or 24 bit images\n");
        ::GlobalUnlock((HGLOBAL) m_hSnowyDIB);
        return;
    }

    if (m_BitsPerPixel == 8 || m_BitsPerPixel == 24)
    {
        CoxKey coxKey(m_pParams->GetKey(), (BITMAPINFO *) lpSnowyDIBHdr,
            hpSnowyDIBBits);

        ::GlobalUnlock((HGLOBAL) m_hSnowyDIB);
    }

    // Sign()
    // This is the function which calls upon the core signing algorithms.
    // WARNING: CURRENTLY THIS FUNCTION ASSUMES THAT WE ALWAYS ARE SIGNING
    // THE "ORIGINAL IMAGE" DIB. THIS MAY BE A BUG.
    // First shot at a function which calls the signer core algorithms
    void CDbDoc::Sign(void)
    {
        long num_pixels, num_colors;
        DWORD image_byte;
        HPSTR src_data, dest_data; // Huge ptrs for copying the image.
        float rms;
        int num_channels;

        HDIB hOriginalDIB = GetOriginalHDIB();
        if (hOriginalDIB == NULL)
            return;

        // Create space for the signed image DIB.
        m_hSignedDIB = (HDIB)::GlobalAlloc(GMEM_MOVEABLE | GMEM_ZEROINIT, m_dwTotalDIBSize);
        if (m_hSignedDIB == 0)
        {
            MessageBox(NULL,
                "Insufficient memory is available for the signed image",
                "Digitarc Signer Warning",
                MB_ICONINFORMATION | MB_OK);
            return;
        }

        // Create Image objects for the images. Note that this locks them in memory.
        Image snowImage(m_hSnowyDIB);
        Image unsignedImage(m_hOriginalDIB);

        // This is ugly, but I have to copy the DIB header stuff into the signed DIB
        // before I can create the signedImage object.
        dest_data = (char *) ::GlobalLock((HGLOBAL) m_hSignedDIB);

        // We want to copy the BITMAPINFO structure from the unsigned to the signed DIB
        src_data = unsignedImage.GetlpDIB();

        // Copy the BITMAPINFOHEADER and palette to the signed DIB space, byte by byte.
        for (image_byte = 0; image_byte < unsignedImage.GetSizeofHeader(); image_byte++)
        {
            *dest_data++ = *src_data++;
        }

        ::GlobalUnlock((HGLOBAL) m_hSignedDIB);

        // Now create the signedImage object, which will lock the DIB in memory again.
        Image signedImage(m_hSignedDIB);

        // For each, create a "byte-wise" packed data array from the DIB 4-byte packing
        snowImage.MakePackedData(FORCE_TO_1_CHANNEL); // snow image always 1 chan
        unsignedImage.MakePackedData();
        signedImage.MakePackedData();

        num_pixels = (long) unsignedImage.GetXDIm() * unsignedImage.GetYDim();
        num_colors = unsignedImage.GetNumColors();

        if (m_BitsPerPixel != 8 && m_BitsPerPixel != 24)
        {
            TRACE("At this time, only sign 8 and 24 bit images\n");
            return;
        }

        // Create and load the luminance scaling look up table.

```

```

float *luminance_lut = new float[256];
::load_luminance_lut(luminance_lut, m_pParams->GetGamma());

// Create and load the key look up table.
char *key_lut = new char[256];
rms = ::load_key_lut(key_lut, m_pParams->GetGain());

long data_length = unsignedImage.GetXDim() * unsignedImage.GetYDim();

// Create a packed msg (will be a user input in future).
if (m_pPackedMsg != NULL)
    delete m_pPackedMsg;
m_pPackedMsg = new PackedMsg( (const char *) m_pParams->GetMessage());

// Set up some arguments and call the core signer.
int x_dim = unsignedImage.GetXDim();
int y_dim = unsignedImage.GetYDim();

if (unsignedImage.GetBitsPerPixel() == 8)
    num_channels = 1;
else if (unsignedImage.GetBitsPerPixel() == 24)
    num_channels = 3;

// const float lut scale = (float)1.0; // Later this will be user controlled.
float *detail_lut = new float[DETAIL_TOTAL];
::load_detail_lut(detail_lut, m_pParams->GetLutScale());

::sign_8bit_single_channel_or_color(unsignedImage.GetPackedData(),
    data_length,
    x_dim,
    y_dim,
    m_pPackedMsg->getMsgBitArray(),
    m_pPackedMsg->getMsgBitArrayLength(),
    snowImage.GetPackedData(),
    data_length,
    key_lut,
    luminance_lut,
    detail_lut,
    STANDARD,
    num_channels,
    m_pParams->GetBumpSize());

delete [] detail_lut;

// Set the timestamp indicating when we signed this puppy.
m_pParams->UpdateSignTime();

delete [] luminance_lut;
delete [] key_lut;

// Now unpack the data in the Image object, back into the standard DIB format
signedImage.UnpackData();

////////////////////////////////////
// Read()
// The read function is the interface to the core recognition algorithm.
// It sets up the necessary data structures needed by the core routine
// and makes the call.
void CDibDoc::Read(HDIB hSignedDIB, BOOL use_super_reader)
{
    long num_pixels, num_colors;
    int num_channels;
    int reading_mode;

    // Create Image objects for the images. Note that this locks them in memory.
    Image snowImage(m_hSnowDIB);
    Image signedImage(hSignedDIB);

    // Create a "byte-wise" packed data array from the DIB 4-byte packing
    signedImage.MakePackedData();
    snowImage.MakePackedData(FORCE_TO_1_CHANNEL); // Snowy images always 1 ch.
    // unsignedImage.MakePackedData();

    num_pixels = (long) signedImage.GetXDim() * signedImage.GetYDim();
    num_colors = signedImage.GetNumColors();

    if (m_BitsPerPixel != 8 && m_BitsPerPixel != 24)
{
TRACE("At this time, only recognize 8 and 24 bit images\n");
return;

// Create and load the luminance scaling look up table.
float *luminance_lut = new float[256];
::load_luminance_lut(luminance_lut, m_pParams->GetGamma());

// Create and load the key look up table.
char *key_lut = new char[256];
::load_key_lut(key_lut, m_pParams->GetGain());

// Create and load the detail look up table.
float *detail_lut = new float[DETAIL_TOTAL];
const float lut_scale = (float)1.0; // Later this will be user controlled.
::load_detail_lut(detail_lut, m_pParams->GetLutScale());

// Determine which bit array to use for the reader's "crude metric"
// computation. If we have just signed this image, then use the
// true message bit array. Otherwise, we are trying to read
// without knowing the the true message, and use the estimated
// message for computation of the metric.
unsigned char *referenceBitArray;
if (m_state == IMAGE_SIGNED || m_state == IMAGE_SIGNED_AND_VERIFIED ||
    m_state == IMAGE_SIGNED_AND_SAVED)
    referenceBitArray = m_pPackedMsg->getMsgBitArray();
else
    referenceBitArray = m_pPackedMsg->getReaderBitArray();

long data_length = signedImage.GetXDim() * signedImage.GetYDim();
long x_offset = 0;
long y_offset = 0;
int x_dim = signedImage.GetXDim();
int y_dim = signedImage.GetYDim();

if (signedImage.GetBitsPerPixel() == 8)
    num_channels = 1;
else if (signedImage.GetBitsPerPixel() == 24)
    num_channels = 3;

// See if we should use the super reader.
if (use_super_reader)
    reading_mode = 1;
else reading_mode = 0;

// Call the core recognizer
::read_8bit_single_channel_or_color(
    signedImage.GetPackedData(),
    x_dim,
    y_dim,
    x_offset,
    y_offset,
    x_dim,
    y_dim,
    m_pPackedMsg->getMsgBitArrayLength(),
    snowImage.GetPackedData(),
    data_length,
    key_lut,
    luminance_lut,
    detail_lut,
    NULL, // No thumbnail at this time
    //unsignedImage.GetPackedData(),
    NULL, // Don't pass original data now
    (const unsigned char *) referenceBitArray,
    &m_crude_metric,
    &m_range,
    m_pPackedMsg->getReaderBitArray(),
    num_channels,
    reading_mode,
    m_pParams->GetBumpSize());

// Convert the recovered message bits back to an ASCII string.
m_pPackedMsg->BitsToString();

TRACE ("The recognizer detected the following string: %s\n",
    m_pPackedMsg->getRecoveredAsciiMsg());

delete [] luminance_lut;
delete [] key_lut;
delete [] detail_lut;

}
////////////////////////////////////
// CDibDoc commands

```



```

        himageToReadDIB = m_hOriginalDIB;
    else if (view_type == SIGNED_VIEW)
        himageToReadDIB = m_hSignedDIB;
    else if (view_type == ALIGNED_VIEW)
        himageToReadDIB = m_pAlignedImage->GetHDIIB();
    else
    {
        MessageBox(NULL, "Bug in OnSettingsReader!", "Error", MB_OK);
        return;
    }

    // Initialize the dialog data
    dlg.m_user_key = m_pParams->GetKey();
    old_key = m_pParams->GetKey();
    dlg.m_msg_length = m_pParams->GetMessage().GetLength();
    dlg.m_gain = m_pParams->GetGain();
    dlg.m_bump_size = m_pParams->GetBumpSize();
    dlg.m_detail_lut_scale = m_pParams->GetLutScale();
    // dlg.m_use_super_reader = m_pParams->GetSuperReaderFlag();

    // Invoke the dialog box
    if (dlg.DoModal() == IDOK)
    {
        m_pParams->SetGain(dlg.m_gain);
        m_pParams->SetBumpSize(dlg.m_bump_size);
        m_pParams->SetLutScale(dlg.m_detail_lut_scale);
        // m_pParams->SetSuperReaderFlag(dlg.m_use_super_reader);

        // If signer has not yet been used, or length changes, need a msg.
        if (m_pParams->GetMessage().GetLength() != (int) dlg.m_msg_length)
        {
            // Create a dummy msg of all x's.
            CString dummy_msg = CString('x', dlg.m_msg_length);
            m_pParams->SetMessage(dummy_msg);
        }

        // Create a PackedMsg object w/ our dummy msg.
        if (m_pPackedMsg != NULL)
            delete m_pPackedMsg;
        m_pPackedMsg = new PackedMsg( (const char *) m_pParams->GetMessage());

        if (dlg.m_user_key != old_key)
        {
            m_pParams->SetKey(dlg.m_user_key);
            new_user_key = TRUE;
        }

        // This is going to take awhile
        BeginWaitCursor();

        // If the user seed has changed, or if we haven't yet created
        // a context-sensitive key, create a snow image.
        if (new_user_key || m_hSnowDIB == NULL)
            MakeSnow(himageToReadDIB);

        // Run the reader and attempt to recover message, and compute metrics.
        Read(himageToReadDIB, m_pParams->GetSuperReaderFlag());

        // Make the state transition: depends on which image was read.
        if (view_type == ORIGINAL_VIEW || view_type == ALIGNED_VIEW)
            m_state = SUSPECT_READ;
        else if (view_type == SIGNED_VIEW)
        {
            if (m_state != IMAGE_SIGNED_AND_SAVED)
                m_state = IMAGE_SIGNED_AND_VERIFIED;
        }

        // WHY? 11/24
        // m_pParams->UpdateSignTime();

        // Now see if a "status image" view exists. If not, create it.
        CDialog *p_statusView;
        p_statusView = (CDialog *) CreateUniqueView(STATUS_VIEW);
        EndWaitCursor();

        // Refresh all of the views (Don't actually need to refresh Original one)
        p_statusView->DoResize();
        UpdateAllViews(NULL);

        // See if the checksum read and the checksum computed from the
        // read message string agree. If not, warn user.
        if (m_pPackedMsg->GetReaderChecksum() !=
            m_pPackedMsg->GetComputedReaderChecksum())
        {
            MessageBox(NULL,
                "The embedded checksum didn't match the computed checksum.",
                "Warning", MB_OK);
        }
    }
}

((CDialog *)pView)->SetViewType(new_type);
return pView;
}

// We get here only if we failed to find a view of "old_type"
return NULL;
}

// OnSettingsAutoPrint()
//
// When the user toggles the "Auto-print Report" item in
// the Options menu, this function is invoked. It simply
// toggles the corresponding member variable.
//
void CDialog::OnSettingsAutoPrint()
{
    if (m_autoPrint == TRUE)
        m_autoPrint = FALSE;
    else
        m_autoPrint = TRUE;
}

// OnUpdateSettingsAutoPrint()
//
// The framework calls this function whenever it is about
// to display the pulldown menu containing the AutoPrint
// Report option. Based on our internal state variable
// m_autoPrint, we set or clear the check mark next to
// the menu item using the pCmdUI->SetCheck() function.
//
void CDialog::OnUpdateSettingsAutoPrint(CCmdUI *pCmdUI)
{
    // Set or clear the check mark in the menu
    if (m_autoPrint == TRUE)
        pCmdUI->SetCheck(TRUE);
    else
        pCmdUI->SetCheck(FALSE);
}

// OnSettingsReader()
//
// Invoked when the user selects the Controls->Reader...
// menu option. Presents a ReadParamsDlg dialog object, and
// deals with the operators inputs. On OK, the Read() function
// is called to use the current parameters and run the recog-
// nition core algorithms to try to detect an embedded
// digimarc message.
//
void CDialog::OnSettingsReader()
{
    ReadDlg dlg;
    CRect rect;
    unsigned old_key;
    BOOL new_user_key = FALSE;
    int view_type;
    HDIB himageToReadDIB;

    // Check to see if we are in a legal state for reading.
    if (m_state == NO_IMAGE)
    {
        MessageBox(NULL,
            "An 8 or 24 bit image must be loaded before using the Reader.",
            "Digimarc Signer Warning",
            MB_ICONINFORMATION | MB_OK);
        return;
    }

    // Determine the type of the active window
    view_type = GetActiveViewType();

    // If active window is not acceptable for reading, warn user & return
    if (view_type != ORIGINAL_VIEW &&
        view_type != SIGNED_VIEW &&
        view_type != ALIGNED_VIEW)
    {
        MessageBox(NULL,
            "The active window must contain an image to be read.",
            "Warning",
            MB_ICONINFORMATION | MB_OK);
        return;
    }

    // Set pointer to the image which is to be read.
    if (view_type == ORIGINAL_VIEW)

```

```

    }
}

// GetActiveViewType()
// Find the active view, determine its type, and return
// it to the caller. The type is one of those listed
// in the DIBview.h file.
// int CDbDoc::GetActiveViewType(void)
{
    BOOL view_found = FALSE;
    POSITION pos = GetFirstViewPosition();
    CView* pView;
    while (pos != NULL)
    {
        pView = GetNextView( pos );

        // If we find it, we return the pointer and we're done.
        if ( ((CDibView*)pView)->IsViewActive() == TRUE)
            return ((CDibView*)pView)->GetViewType();
    }

    // We can get here when other apps are running and Windows sends message
    // resulting in CDbDoc::OnUpdateFileSaveAs() being called.
    // MessageBox(NULL, "Error in GetActiveViewType!", "Error", MB_OK);
    return(UNKNOWN_VIEW);
}

// GetActiveView()
// Return a pointer to the active view (i.e., a CDibView* ), or NULL
// if something goes wrong.
// CDibView* CDbDoc::GetActiveView(void)
{
    BOOL view_found = FALSE;
    POSITION pos = GetFirstViewPosition();
    CView* pView;
    while (pos != NULL)
    {
        pView = GetNextView( pos );

        // If we find it, we return the pointer and we're done.
        if ( ((CDibView*)pView)->IsViewActive() == TRUE)
            return (CDibView*)pView;
    }

    // We can get here when other apps are running and Windows sends message
    // resulting in CDbDoc::OnUpdateFileSaveAs() being called.
    // MessageBox(NULL, "Error in GetActiveViewType!", "Error", MB_OK);
    return(NULL);
}

// OnSettingsAutoread()
// When the user toggles the "Auto-read after Signing" item in
// the Options menu, this function is invoked. It simply
// toggles the corresponding member variable.
// We currently also toggle the application level variable,
// so that the settings are global to all docs.
// void CDbDoc::OnSettingsAutoread()
{
    if (m_autoread == TRUE)
    {
        m_autoread = FALSE;
        ((CDibLookApp *)AfxGetApp())->m_autoread = FALSE;
    }
    else
    {
        m_autoread = TRUE;
        ((CDibLookApp *)AfxGetApp())->m_autoread = TRUE;
    }
}

// OnUpdateSettingsAutoread()
// The framework calls this function whenever it is about
// to display the pulldown menu containing the Autoread
// option. Based on our internal state variable

```

```

// m_autoread, we set or clear the check mark next to
// the menu item using the pCmdUI->SetCheck() function.
// void CDbDoc::OnUpdateSettingsAutoread(CCmdUI* pCmdUI)
{
    // Set or clear the check mark in the menu
    if ( ((CDibLookApp *)AfxGetApp())->m_autoread == TRUE)
        pCmdUI->SetCheck(TRUE);
    else
        pCmdUI->SetCheck(FALSE);
}

// OnSettingsAlign()
// This function is called when the user selects the "Align" menu option.
// A CFileDialog object is created and used in order for the operator
// to specify the name of the "Reference Image" (a signed or unsigned
// original image used as the template).
// void CDbDoc::OnSettingsAlign()
{
    CString refname;
    BOOL success_flag;

    // Create a filter for the types of files the file dialog will offer
    char szFilter[] =
        "Windows Bit Map Files (*.bmp)|*.bmp|*.bmp|Device Independent Bitmaps (*.dib)|*.dib|"
        "All Files (*.*)|*.*||";

    // Construct a file dialog
    CFileDialog
        fileDlg(TRUE,
            "*.BMP",
            NULL,
            OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
            szFilter);

    // Over-ride the default title in the file dialog window
    fileDlg.m_ofn.lpstrTitle = "Select a template file to be used for alignment";

    // Display the file dialog
    if (fileDlg.DoModal() == IDOK)
    {
        // Get the name of the reference image file.
        refname = fileDlg.GetPathName();

        BeginWaitCursor();

        // Create an image object for the reference image.
        // (If one already exists, delete it first).
        if (m_pRefImage != NULL)
            delete m_pRefImage;
        m_pRefImage = new Image(refname);

        if (m_pRefImage->GetFileOK == FALSE) // bail out if something went wrong
            return;

        // Display the reference image
        CreateUniqueView(REF_VIEW);

        // Update all views (NULL);

        TRACE("Call the Align() function (this is a test of trace output.)\n");

        // Do the actual alignment and change update the state description.
        success_flag = Align_it();

        if (success_flag)
        {
            m_state = SUSPECT_ALIGNED;

            // Now, the template image object has had its packed data array replaced
            // by the aligned, co-extensive image. Need to move this packed data
            // into the DIB array for display (and possible file saving) purposes.
            m_pRefImage->UnpackData();

            // We now call the image the Aligned image, not reference
            m_pAlignedImage = m_pRefImage;
            m_pRefImage = NULL;

            CreateUniqueView(ALIGNED_VIEW);

            // Create a status view, if it doesn't already exist.
            CDbView *p_statusView;
            p_statusView = (CDibView *) CreateUniqueView(STATUS_VIEW);

            p_statusView->DoResize();

            UpdateAllViews(NULL);

```



```

void InitDIBData();
// Implementation
protected:
virtual ~CDibDoc();

virtual BOOL OnSaveDocument(const char* pszPathName);
virtual BOOL OnOpenDocument(const char* pszPathName);
//void OnEditSettings();

private:
void MangleDIB(void);
void CdbDoc:DumpBitmapInfoHeader() const;
void MakeSnow(HDIB hParentDIB);
void Sign(void);
void Read(HDIB hSignedDIB, BOOL use_super_reader);
BOOL Align_it(void);
CView* CreateUniqueView(int view_type);
CView* ChangeViewType(int old_type, int new_type);
int GetActiveViewType(void);

CDibView *GetActiveView(void);

int m_state;
CString m_filename;
float m_crude_metric;
float m_range;

Image *m_pRefImage;
Image *m_pAlignedImage;

Align *m_pAlign;

protected:
// HDIB m_hDIB; // Obsolete
CPalette* m_palDIB;
CSize m_sizeDoc;
int m_BitsPerPixel;

CView *m_pSignedView;

// Ptr to the initially loaded image, unmodified by signing.
HDIB m_hOriginalDIB;

// Add additional DIB handles for the snowy image and signed image.
HDIB m_hSnowyDIB;
HDIB m_hSignedDIB;

// Need to know total space needed for these guys.
DWORD m_dwTotalDIBSize;

// Pointer to parameters object.
SignerParams *m_pParams;

PackedMsg *m_pPackedMsg;

BOOL m_autoprint;
BOOL m_autoread;

#ifdef _DEBUG
virtual void AssertValid() const;
virtual void Dump(CDumpContext& dc) const;
#endif
protected:
virtual BOOL OnNewDocument();

// Generated message map functions
protected:
//{{AFX_MSG(CDibDoc)
afx_msg void OnSettingsSigner();
afx_msg void OnSettingsAutoprint();
afx_msg void OnUpdateSettingsAutoprint(CCmdUI* pCmdUI);
afx_msg void OnSettingsReader();
afx_msg void OnSettingsAutoread();
afx_msg void OnUpdateSettingsAutoread(CCmdUI* pCmdUI);
afx_msg void OnSettingsAlign();
afx_msg void OnUpdateFileSaveAs(CCmdUI* pCmdUI);
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

// sign.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "signer.h"

#include "mainfrm.h"
#include "signdoc.h"
#include "signview.h"

#include "mychildw.h"

// #include "AFXPRIV.H"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

char *global_cmd_line_args;
///////////////////////////////////////////////////
// CDibLookApp
BEGIN_MESSAGE_MAP(CDibLookApp, CWinApp)
//{{AFX_MSG_MAP(CDibLookApp)
ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
//}}AFX_MSG_MAP
// Standard file based document commands
ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
// Standard print setup command
ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

///////////////////////////////////////////////////
// Place all significant initialization in InitInstance
CDibLookApp::CDibLookApp()
{
    m_lpParams = NULL;
    m_autoread = FALSE;
}

CDibLookApp::~CDibLookApp()
{
    if (m_lpParams != NULL)
        delete m_lpParams;
}

// The one and only CDibLookApp object
///////////////////////////////////////////////////
// CDibLookApp initialization
// CDibLookApp initialization
// CDibLookApp::InitInstance()
{
    // Standard initialization
    // (if you are not using these features and wish to reduce the size
    // of your final executable, you should remove the following initialization
    // SetDialogBKColor(); // set dialog background color
    LoadStdProfileSettings(); // Load standard INI file options (including MRU)

    // Register document templates which serve as connection between
    // documents and views. Views are contained in the specified view
    AddDocTemplate(new CMultiDocTemplate(IDR_DIBTYPE,
        RUNTIME_CLASS(CDibDoc),
        RUNTIME_CLASS(CMyChildWnd), // I replace CMDIChildWnd
        RUNTIME_CLASS(CDibView)));

    // create main MDI Frame window
    CMainFrame* pMainFrame = new CMainFrame;
    if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
        return FALSE;
    pMainFrame->ShowWindow(m_nCmdShow);
    pMainFrame->UpdateWindow();
    m_pMainWnd = pMainFrame;

    // enable file manager drag/drop and DDE Execute open
    m_pMainWnd->DragAcceptFiles();
    EnableShellOpen();
    RegisterShellFileTypes();
}

```

```

// As a test, save a global copy of command line args
// Global_end_line_args = m_lpCmdLine;
m_lpParams = new SignerParams(m_lpCmdLine);

// DEBUG: display the command line before we parse it.
// ArMessageBox(m_lpCmdLine);

// simple command line parsing
if (m_lpParams->GetInputFilename() == NULL)
{
    // create a new (empty) document
    // OnFileNew();
}
else if ((m_lpCmdLine[0] == '-' || m_lpCmdLine[0] == '/') &&
(m_lpCmdLine[1] == 'e' || m_lpCmdLine[1] == 'g'))
{
    // program launched embedded - wait for DDE or OLE open
}
else
{
    // open an existing document
    OpenDocumentFile(m_lpParams->GetInputFilename());
}

// Try adding another window.
// pMainFrame->OnWindowNew(); fails: this is a protected member.
// pMainFrame->SendMessage(ID_WINDOW_NEW);
// pMainFrame->MyOnWindowNewTest();

return TRUE;
}

////////////////////////////////////
// CaboutDlg dialog used for App About
////////////////////////////////////

class CaboutDlg : public CDialog
{
public:
    CaboutDlg() : CDialog(CAboutDlg::IDD)
    {
        //{{AFX_DATA_INIT(CaboutDlg)
        //}}AFX_DATA_INIT
    }

    // Dialog Data
    enum { IDD = IDD_ABOUTBOX };

    // Implementation
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //{{AFX_MSG(CaboutDlg)
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

void CaboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CaboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CaboutDlg, CDialog)
    //{{AFX_MSG_MAP(CaboutDlg)
    //}}AFX_MSG_MAP
    END_MESSAGE_MAP()

// App command to run the dialog
void CblookApp::OnAppAbout()
{
    CaboutDlg aboutDlg;
    aboutDlg.DoModal();
}

////////////////////////////////////
// CDblookApp commands
////////////////////////////////////

// signer.h : main header file for the SIGNER application
//
// error include 'stdafx.h' before including this file for PCH
//endif _AFXWIN_H_
//endif
#include "resource.h" // main symbols
#include "params.h"
#define WM_DOREALIZE (WM_USER + 0)
////////////////////////////////////
// CDblookApp:
// See dlook.cpp for the implementation of this class
////////////////////////////////////
class CDblookApp : public CWinApp
{
public:
    CDblookApp();
    ~CDblookApp();

    // Create a command line parameter object.
    SignerParams *m_lpParams;
    SignerParams *GetParams(void) {return m_lpParams;}

    BOOL m_autoread;

    // Overrides
    virtual BOOL InitInstance();

    // Implementation
    //{{AFX_MSG(CDblookApp)
    afx_msg void OnAppAbout();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////
// Microsoft Developer Studio generated resource script.
//include "resource.h"

#define APSTUDIO_READONLY_SYMBOLS
////////////////////////////////////
// Generated from the TEXTINCLUDE 2 resource.
//include "afxres.h"
////////////////////////////////////
#undef APSTUDIO_READONLY_SYMBOLS
// English (U.S.) resources
//if defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#define _WIN32
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
#pragma code_page(1252)
#endif _WIN32

#ifdef APSTUDIO_INVOKED
// TEXTINCLUDE
//
1 TEXTINCLUDE DISCARDABLE
BEGIN
    "resource.h\0"
END
2 TEXTINCLUDE DISCARDABLE
BEGIN
    "#include \"afxres.h\"\\r\\n"
    "\\0"
END
3 TEXTINCLUDE DISCARDABLE
BEGIN
    "#include \"afxres.rc\"\\r\\n"
    "#include \"afxprint.rc\"\\r\\n"
    "\\0"
END

```



```

"\0"
END
#endif // APSTUDIO_INVOKED
////////////////////////////////////
// Icon
//
// Icon with lowest ID value placed first to ensure application icon
// remains consistent on all systems.
IDR_MAINFRAME ICON DISCARDABLE "RES\DILOOK.ICO"
IDR_DIBTYPE ICON DISCARDABLE "RES\DIIDDOC.ICO"
////////////////////////////////////
// Bitmap
//
IDR_MAINFRAME BITMAP MOVEABLE PURE "RES\TOOLBAR.BMP"
////////////////////////////////////
// Menu
//
IDR_MAINFRAME MENU PRELOAD DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&New\Ctrl+N" ID_FILE_NEW
        MENUITEM "&Open...\tCtrl+O", ID_FILE_OPEN
        MENUITEM SEPARATOR
        MENUITEM "&Print Setup...", ID_FILE_PRINT_SETUP
        MENUITEM SEPARATOR
        MENUITEM "&Recent File", ID_FILE_MRU_FILE1, GRAYED
        MENUITEM SEPARATOR
        MENUITEM "&Exit", ID_APP_EXIT
    END
    POPUP "&View"
    BEGIN
        MENUITEM "&toolbar", ID_VIEW_TOOLBAR
        MENUITEM "&status Bar", ID_VIEW_STATUS_BAR
    END
    POPUP "&help"
    BEGIN
        MENUITEM "&about SIGNER...", ID_APP_ABOUT
    END
    MENUITEM "&about SIGNER...", ID_APP_ABOUT
END

////////////////////////////////////
// Dialog
//
IDD_ABOUTBOX_DIALOG DISCARDABLE 34, 22, 216, 91
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
FONT 8, "MS Sans Serif"
BEGIN
    ICON IDR_MAINFRAME IDC_STATIC,11,17,18,20
    LTEXT "Digimarc Win32 Signer version 0.24", IDC_STATIC,40,10,127,8
    LTEXT "Copyright - 1995, 1996", IDC_STATIC,40,119,8
    DEFPUSHBUTTON "OK", IDOK,16,6,32,14,WS_GROUP
    LTEXT "For internal evaluation only.", IDC_STATIC,40,55,100,10
    LTEXT "Rev 04/10/96", IDC_STATIC,40,25,57,8
END

IDR_PARAMS_DIALOG_DIALOG DISCARDABLE 0, 0, 232, 179
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON "OK", IDOK,45,144,50,14
    PUSHBUTTON "Cancel", IDCANCEL,135,144,50,14
    IDC_MESSAGE,6,17,221,15,ES_AUTOHSCROLL
    LTEXT "Key:", IDC_STATIC,8,48,30,8
    IDC_EDIT_KEY,92,45,40,13,ES_AUTOHSCROLL
    LTEXT "Gain:", IDC_STATIC,8,70,30,9
    IDC_EDIT_GAIN,92,67,40,13,ES_AUTOHSCROLL
    LTEXT "Bump Size:", IDC_STATIC,8,93,44,8
    IDC_BUMP_SIZE,92,89,40,13,ES_AUTOHSCROLL
    LTEXT "Message:", IDC_MESSAGE,6,5,58,10
    LTEXT "Detail Gain:", IDC_STATIC,8,115,60,8
    IDC_DETAIL_SCALE,92,111,40,14,ES_AUTOHSCROLL
END

IDD_READ_DIALOG_DIALOG DISCARDABLE 0, 0, 152, 200
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Reader Controls Dialog"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON "OK", IDOK,8,160,50,15

```

[illegible]


```

-erase "Debug\vc40.idb"
-erase "Debug\SignerWin32.bsc"
-erase "Debug\Dibapi.sbr"
-erase "Debug\Readlg.sbr"
-erase "Debug\Myfile.sbr"
-erase "Debug\Mychildw.sbr"
-erase "Debug\Coxkey.sbr"
-erase "Debug\Signview.sbr"
-erase "Debug\Signer.sbr"
-erase "Debug\Stdafx.sbr"
-erase "Debug\Read.sbr"
-erase "Debug\Packmsg.sbr"
-erase "Debug\Fft.sbr"
-erase "Debug\Sign.sbr"
-erase "Debug\Image.sbr"
-erase "Debug\Parmadlg.sbr"
-erase "Debug>Mainfrm.sbr"
-erase "Debug\Signdoc.sbr"
-erase "Debug\Align.sbr"
-erase "Debug\Params.sbr"
-erase "Debug\SignerWin32.exe"
-erase "Debug\Params.obj"
-erase "Debug\Dibapi.obj"
-erase "Debug\Readlg.obj"
-erase "Debug\Myfile.obj"
-erase "Debug\Mychildw.obj"
-erase "Debug\Coxkey.obj"
-erase "Debug\Signview.obj"
-erase "Debug\Signer.obj"
-erase "Debug\Stdafx.obj"
-erase "Debug\Read.obj"
-erase "Debug\Packmsg.obj"
-erase "Debug\Fft.obj"
-erase "Debug\Sign.obj"
-erase "Debug\Image.obj"
-erase "Debug\Parmadlg.obj"
-erase "Debug>Mainfrm.obj"
-erase "Debug\Signdoc.obj"
-erase "Debug\Align.obj"
-erase "Debug\Signer.res"

*$(OUTDIR) :
if not exist "$(OUTDIR)/$(NULL)" mkdir "$(OUTDIR)"

# ADD BASE CPP /nologo /MTd /W3 /Gm /GX /ZI /Od /D "WIN32" /D "_DEBUG" /D "_WINDOWS" /D "_MBCS" /PR /YX /c
# ADD CPP /nologo /MTd /W3 /Gm /GX /ZI /Od /D "WIN32" /D "_DEBUG" /D "_WINDOWS" /D "_MBCS" /PR /YX /c
CPP_PROJ=/nologo /MTd /W3 /Gm /GX /ZI /Od /D "WIN32" /D "_DEBUG" /D "_WINDOWS" /
/D "_MBCS" /PR "$(INTDIR)" /Fp "$(INTDIR)\SignerWin32.pch" /YX /Fo "$(INTDIR)\"
/Fd "$(INTDIR)" /c
CPP_OBJS=/Debug/
CPP_SBRS=/Debug/
# ADD BASE MTL /nologo /D "DEBUG" /win32
# ADD MTL /nologo /D "DEBUG" /win32
MTL_PROJ=/nologo /D "DEBUG" /win32
# ADD BASE RSC /I /Ox409 /d "DEBUG"
# ADD RSC /I /Ox409 /d "DEBUG"
RSC_PROJ=/I /Ox409 /fo "$(INTDIR)\Signer.res" /d "_DEBUG"
BSC32=/BSC32.exe
# ADD BASE BSC32 /nologo
# ADD BSC32 /nologo
BSC32_FLAGS=/nologo /o "$(OUTDIR)\SignerWin32.bsc"
BSC32_SBRS=
$(INTDIR)\Dibapi.sbr"
$(INTDIR)\Readlg.sbr"
$(INTDIR)\Myfile.sbr"
$(INTDIR)\Mychildw.sbr"
$(INTDIR)\Coxkey.sbr"
$(INTDIR)\Signview.sbr"
$(INTDIR)\Signer.sbr"
$(INTDIR)\Stdafx.sbr"
$(INTDIR)\Read.sbr"
$(INTDIR)\Packmsg.sbr"
$(INTDIR)\Fft.sbr"
$(INTDIR)\Sign.sbr"
$(INTDIR)\Image.sbr"
$(INTDIR)\Parmadlg.sbr"
$(INTDIR)\Mainfrm.sbr"
$(INTDIR)\Signdoc.sbr"
$(INTDIR)\Align.sbr"
$(INTDIR)\Params.sbr"
*$(OUTDIR)\SignerWin32.bsc" : *$(OUTDIR) " $(BSC32_SBRS)
$(BSC32) @<
$(BSC32_FLAGS) $(BSC32_SBRS)
<<

LINK32=/link.exe
# ADD BASE LINK32 oldnames.lib /nologo /stack:0x2800 /subsystem:windows /debug /machine:ix86
# ADD LINK32 oldnames.lib /nologo /stack:0x2800 /subsystem:windows /profile /debug /machine:ix86
LINK32_FLAGS=oldnames.lib /nologo /stack:0x2800 /subsystem:windows /profile /
debug /machine:ix86 /def:"\Signer.def" /out:"$(OUTDIR)\SignerWin32.exe"
DEP_FILE=
LINK32_OBJS=
$(INTDIR)\Params.obj"
$(INTDIR)\Dibapi.obj"
$(INTDIR)\Readlg.obj"
$(INTDIR)\Myfile.obj"
$(INTDIR)\Mychildw.obj"
$(INTDIR)\Coxkey.obj"
$(INTDIR)\Signview.obj"
$(INTDIR)\Signer.obj"
$(INTDIR)\Stdafx.obj"
$(INTDIR)\Read.obj"
$(INTDIR)\Packmsg.obj"
$(INTDIR)\Fft.obj"
$(INTDIR)\Sign.obj"
$(INTDIR)\Image.obj"
$(INTDIR)\Parmadlg.obj"
$(INTDIR)\Mainfrm.obj"
$(INTDIR)\Signdoc.obj"
$(INTDIR)\Align.obj"
$(INTDIR)\Signer.res"
*$(OUTDIR)\SignerWin32.exe" : *$(OUTDIR) " $(DEP_FILE) $(LINK32_OBJS)
$(LINK32) @<
$(LINK32_FLAGS) $(LINK32_OBJS)
<<
ENDIF

.c $(CPP_OBJS) .obj :
$(CPP) $(CPP_PROJ) <
.cpp $(CPP_OBJS) .obj :
$(CPP) $(CPP_PROJ) <
.cxx $(CPP_OBJS) .obj :
$(CPP) $(CPP_PROJ) <
.c $(CPP_SBRS) .sbr :
$(CPP) $(CPP_PROJ) <
.cpp $(CPP_SBRS) .sbr :
$(CPP) $(CPP_PROJ) <
.cxx $(CPP_SBRS) .sbr :
$(CPP) $(CPP_PROJ) <
#####
# Begin Target
# Name "Signer - Win32 Release"
# Name "Signer - Win32 Debug"
IF "$(CFG)" == "Signer - Win32 Release"
!ELSEIF "$(CFG)" == "Signer - Win32 Debug"
ENDIF
#####
#####
SOURCE=.\Coxkey.cpp
DEP_CPP_COXKE=
".\Coxkey.h"
".\Dibapi.h"
".\Stdafx.h"
*$(INTDIR)\Coxkey.obj" : $(SOURCE) $(DEP_CPP_COXKE) "$(INTDIR) "
*$(INTDIR)\Coxkey.sbr" : $(SOURCE) $(DEP_CPP_COXKE) "$(INTDIR) "
# End Source File
#####
# Begin Source File
SOURCE=.Dibapi.cpp
DEP_CPP_DIBAP=
".\Stdafx.h"
".\Dibapi.h"
*$(INTDIR)\Dibapi.obj" : $(SOURCE) $(DEP_CPP_DIBAP) "$(INTDIR) "
*$(INTDIR)\Dibapi.sbr" : $(SOURCE) $(DEP_CPP_DIBAP) "$(INTDIR) "

```

```

# End Source File
#####
# Begin Source File

SOURCE=.\Image.cpp
DEP_CPP_IMAGE=\\
"..\Image.h"\\
"..\Dibapi.h"\\
"..\Stdafx.h"\\

"$ (INTDIR)\Image.obj" : $(SOURCE) $(DEP_CPP_IMAGE) "$ (INTDIR)"
"$ (INTDIR)\Image.sbr" : $(SOURCE) $(DEP_CPP_IMAGE) "$ (INTDIR)"

# End Source File
#####
# Begin Source File

SOURCE=.\Mainfrm.cpp
!IF "$(CFG)" == "Signer - Win32 Release"
DEP_CPP_MAINF=\\
"..\Stdafx.h"\\
"..\Signer.h"\\
"..\Mainfrm.h"\\
"..\Params.h"\\

"$ (INTDIR)\Mainfrm.obj" : $(SOURCE) $(DEP_CPP_MAINF) "$ (INTDIR)"
"$ (INTDIR)\Mainfrm.sbr" : $(SOURCE) $(DEP_CPP_MAINF) "$ (INTDIR)"

!ELSEIF "$(CFG)" == "Signer - Win32 Debug"
DEP_CPP_MAINF=\\
"..\Stdafx.h"\\
"..\Signer.h"\\
"..\Mainfrm.h"\\

"$ (INTDIR)\Mainfrm.obj" : $(SOURCE) $(DEP_CPP_MAINF) "$ (INTDIR)"
"$ (INTDIR)\Mainfrm.sbr" : $(SOURCE) $(DEP_CPP_MAINF) "$ (INTDIR)"

!ENDIF

# End Source File
#####
# Begin Source File

SOURCE=.\Myfile.cpp
DEP_CPP_MYFIL=\\
"..\Stdafx.h"\\
"..\Dibapi.h"\\

"$ (INTDIR)\Myfile.obj" : $(SOURCE) $(DEP_CPP_MYFIL) "$ (INTDIR)"
"$ (INTDIR)\Myfile.sbr" : $(SOURCE) $(DEP_CPP_MYFIL) "$ (INTDIR)"

# End Source File
#####
# Begin Source File

SOURCE=.\Packmsg.cpp
DEP_CPP_PACKMG=\\
"..\Stdafx.h"\\
"..\packmsg.h"\\

"$ (INTDIR)\Packmsg.obj" : $(SOURCE) $(DEP_CPP_PACKMG) "$ (INTDIR)"
"$ (INTDIR)\Packmsg.sbr" : $(SOURCE) $(DEP_CPP_PACKMG) "$ (INTDIR)"

# End Source File
#####
# Begin Source File

SOURCE=.\Params.cpp
DEP_CPP_PARAM=\\
"..\Stdafx.h"\\
"..\Params.h"\\

"$ (INTDIR)\Stdafx.obj" : $(SOURCE) $(DEP_CPP_STDAF) "$ (INTDIR)"
"$ (INTDIR)\Stdafx.sbr" : $(SOURCE) $(DEP_CPP_STDAF) "$ (INTDIR)"

# End Source File
#####

```

```

"$ (INTDIR)\Params.obj" : $(SOURCE) $(DEP_CPP_PARAM) "$ (INTDIR)"
"$ (INTDIR)\Params.sbr" : $(SOURCE) $(DEP_CPP_PARAM) "$ (INTDIR)"

# End Source File
#####
# Begin Source File

SOURCE=.\Parmsdlg.cpp
!IF "$(CFG)" == "Signer - Win32 Release"
DEP_CPP_PARMS=\\
"..\Stdafx.h"\\
"..\Signer.h"\\
"..\Parmsdlg.h"\\
"..\Params.h"\\

"$ (INTDIR)\Parmsdlg.obj" : $(SOURCE) $(DEP_CPP_PARMS) "$ (INTDIR)"
"$ (INTDIR)\Parmsdlg.sbr" : $(SOURCE) $(DEP_CPP_PARMS) "$ (INTDIR)"

!ELSEIF "$(CFG)" == "Signer - Win32 Debug"
DEP_CPP_PARMS=\\
"..\Stdafx.h"\\
"..\Signer.h"\\
"..\Parmsdlg.h"\\

"$ (INTDIR)\Parmsdlg.obj" : $(SOURCE) $(DEP_CPP_PARMS) "$ (INTDIR)"
"$ (INTDIR)\Parmsdlg.sbr" : $(SOURCE) $(DEP_CPP_PARMS) "$ (INTDIR)"

!ENDIF

# End Source File
#####
# Begin Source File

SOURCE=.\Read.cpp
DEP_CPP_READ=\\
"..\Read.h"\\
"..\Sign.h"\\
"..\Std.h"\\
"..\Stdafx.h"\\

"$ (INTDIR)\Read.obj" : $(SOURCE) $(DEP_CPP_READ_) "$ (INTDIR)"
"$ (INTDIR)\Read.sbr" : $(SOURCE) $(DEP_CPP_READ_) "$ (INTDIR)"

# End Source File
#####
# Begin Source File

SOURCE=.\Sign.cpp
DEP_CPP_SIGN=\\
"..\Sign.h"\\
"..\Stdafx.h"\\

"$ (INTDIR)\Sign.obj" : $(SOURCE) $(DEP_CPP_SIGN_) "$ (INTDIR)"
"$ (INTDIR)\Sign.sbr" : $(SOURCE) $(DEP_CPP_SIGN_) "$ (INTDIR)"

# End Source File
#####
# Begin Source File

SOURCE=.\Stdafx.cpp
DEP_CPP_STDAF=\\
"..\Stdafx.h"\\

"$ (INTDIR)\Stdafx.obj" : $(SOURCE) $(DEP_CPP_STDAF) "$ (INTDIR)"
"$ (INTDIR)\Stdafx.sbr" : $(SOURCE) $(DEP_CPP_STDAF) "$ (INTDIR)"

# End Source File
#####

```



```

"$(INTDIR)\ReadDlg.obj" : $(SOURCE) $(DEP_CPP_READD) "$(INTDIR)"
"$(INTDIR)\ReadDlg.sbr" : $(SOURCE) $(DEP_CPP_READD) "$(INTDIR)"
ENDIF

# End Source File
#####
# Begin Source File
SOURCE=.\Signer.def

IF "$(CFG)" == "Signer - Win32 Release"
ELSEIF "$(CFG)" == "Signer - Win32 Debug"
ENDIF

# End Source File
#####
# Begin Source File
SOURCE=.\Align.cpp

"$(INTDIR)\Align.obj" : $(SOURCE) "$(INTDIR)"
"$(INTDIR)\Align.sbr" : $(SOURCE) "$(INTDIR)"

# End Source File
#####
# Begin Source File
SOURCE=.\Pft.cpp

"$(INTDIR)\Pft.obj" : $(SOURCE) "$(INTDIR)"
"$(INTDIR)\Pft.sbr" : $(SOURCE) "$(INTDIR)"

# End Source File
#####
# End Target
# End Project
#####
#####

SIGNVIEW.CPP
#####
// Signview.cpp
//
// Implementation of the CDibView class
//
#include "stdafx.h"
#include "signer.h"

#include "signdoc.h"
#include "signview.h"
#include "dibapi.h"
#include "mainfrm.h"
#include "Align.h"

#include <strstream>
#include <omanip>

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

// CDibView
//
IMPLEMENT_DYNCREATE(CDibView, CScrollView)

BEGIN_MESSAGE_MAP(CDibView, CScrollView)
    //({AFX_MSG_MAP(CDibView)
    ON_COMMAND(ID_EDIT_COPY, OnEditCopy)
    ON_UPDATE_COMMAND_UI(ID_EDIT_COPY, OnUpdateEditCopy)
    ON_COMMAND(ID_EDIT_PASTE, OnEditPaste)
    ON_UPDATE_COMMAND_UI(ID_EDIT_PASTE, OnUpdateEditPaste)
    ON_MESSAGE(WM_DOREALIZE, OnDoRealize)
    ON_COMMAND(ID_VIEW_SIGNED, OnViewSigned)
    ON_COMMAND(ID_VIEW_UNSIGNED, OnViewUnsigned)
    ON_COMMAND(ID_VIEW_SNOWY_IMAGE, OnViewSnowyImage)

```

```

ON_COMMAND(ID_VIEW_STATUS, OnViewStatus)
ON_UPDATE_COMMAND_UI(ID_VIEW_SIGNED, OnUpdateViewSigned)
ON_UPDATE_COMMAND_UI(ID_VIEW_SNOWY_IMAGE, OnUpdateViewSnowyImage)
ON_UPDATE_COMMAND_UI(ID_VIEW_STATUS, OnUpdateViewStatus)
//})AFX_MSG_MAP

// Standard printing commands
ON_COMMAND(ID_FILE_PRINT, CScrollView::OnFilePrint)
ON_COMMAND(ID_FILE_PRINT_PREVIEW, CScrollView::OnFilePrintPreview)
END_MESSAGE_MAP()

// CDibView()
//
// The constructor
//
//
// CDibView::CDibView()
//
// m_viewType = ORIGINAL_VIEW; // default type of view
// m_bThisViewActive = FALSE; // View is initially inactive
// m_bDocResizeStatusView = FALSE;
//
//
// -CDibView()
//
// The destructor.
//
//
// CDibView::~CDibView()
//
//
//
// GetHdib()
//
// Returns the Hdib (handle to the DIB) of the current view. Note that
// it doesn't make sense to call this if the current view is the status
// view, or any other view which isn't displaying a DIB.
//
//
// Hdib CDibView::GetHdib(void)
//
// {
//     CDibDoc* pDoc = GetDocument();
//
//     switch (m_viewType)
//     {
//         case ORIGINAL_VIEW:
//             return pDoc->GetOriginalHdib();
//             break;
//         case SIGNED_VIEW:
//             return pDoc->GetSignedHdib();
//             break;
//         case SNOWY_VIEW:
//             return pDoc->GetSnowyHdib();
//             break;
//         case REF_VIEW:
//             return pDoc->GetRefHdib();
//             break;
//         case ALIGNED_VIEW:
//             return pDoc->GetAlignedHdib();
//         case STATUS_VIEW:
//             return
//             default:
//                 return pDoc->GetOriginalHdib();
//                 break;
//     }
// }

//
// OnDraw()
//
// Given a pointer to a CDC (device context), this function is responsible
// for drawing the current view.
//
// void CDibView::OnDraw(CDC* pDC)
//
// {
//     if (m_viewType == STATUS_VIEW)
//     {
//         DisplayStatus(pDC);
//     }
//     else
//     {
//         CDibDoc* pDoc = GetDocument();
//         Hdib Hdib = GetHdib();
//         if (Hdib != NULL)
//         {

```



```

pDoc->InitDIBData(); // set up new size & palette
pDoc->SetModifiedFlag(TRUE);
SetScrollSizes(MM_TEXT, pDoc->GetDocSize());
OnDoRealize((LPARAM)m_hwnd,0); // realize the new palette
pDoc->UpdateAllViews(NULL);
}
EndWaitCursor();
}

////////////////////////////////////
void OnUpdateEditPaste()
{
////////////////////////////////////
void CDbView::OnUpdateEditPaste(CCmdUI* pCmdUI)
{
pCmdUI->Enable(1); // ClipboardFormatAvailable(CF_DIB));
}

////////////////////////////////////
void OnViewSigned()
{
////////////////////////////////////
void CDbView::OnViewSigned()
{
CDbDoc* pDoc = GetDocument();
m_viewType = SIGNED_VIEW;
//pDoc->SetModifiedFlag(TRUE);
// Set the window title.
GetParent()->SetWindowText(GetDocument()->GetTitle() + " -Signed");
pDoc->UpdateAllViews(NULL);
}

////////////////////////////////////
void OnViewUnsigned()
{
////////////////////////////////////
void CDbView::OnViewUnsigned()
{
CDbDoc* pDoc = GetDocument();
m_viewType = ORIGINAL_VIEW;
// Set the window title.
GetParent()->SetWindowText(GetDocument()->GetTitle() + " -Original");
pDoc->UpdateAllViews(NULL);
}

////////////////////////////////////
void OnViewSnowyImage()
{
////////////////////////////////////
void CDbView::OnViewSnowyImage()
{
CDbDoc* pDoc = GetDocument();
m_viewType = SNOWY_VIEW;
// Set the window title.
GetParent()->SetWindowText(GetDocument()->GetTitle() + " -Code Pattern");
pDoc->UpdateAllViews(NULL);
}

////////////////////////////////////
void OnViewStatus()
{
////////////////////////////////////
void CDbView::OnViewStatus()
{
CDbDoc* pDoc = GetDocument();
m_viewType = STATUS_VIEW;
// Set the window title.
GetParent()->SetWindowText(GetDocument()->GetTitle() + " -Status");
pDoc->UpdateAllViews(NULL);
}

////////////////////////////////////
void SetViewType()
{

```

```

////////////////////////////////////
void CDbView::SetViewType(int type)
{
CDbDoc* pDoc = GetDocument();
switch (type)
{
case SIGNED_VIEW:
m_viewType = SIGNED_VIEW;
// Set the window title.
GetParent()->SetWindowText(GetDocument()->GetTitle() + " -Signed");
break;
case REF_VIEW:
m_viewType = REF_VIEW;
// Set the window title.
GetParent()->SetWindowText(GetDocument()->GetTitle() + " -Reference");
break;
case ALIGNED_VIEW:
m_viewType = ALIGNED_VIEW;
// Set the window title.
GetParent()->SetWindowText(GetDocument()->GetTitle() + " -Aligned");
break;
case STATUS_VIEW:
m_viewType = STATUS_VIEW;
// Set the window title.
GetParent()->SetWindowText(GetDocument()->GetTitle() + " -Status");
break;
default:
// This is an error.
// afxmessage
break;
}
}

////////////////////////////////////
void DisplayStatus()
{
////////////////////////////////////
void CDbView::DisplayStatus(CDC *pDC)
{
CDbDoc* pDoc = GetDocument();
TEXTMETRIC tm;
CString text;
CRect rect;
CTime t;

pDC->GetTextMetrics(&tm);
int col = 20*tm.tmAveCharWidth;
int line = tm.tmHeight;
ostrstream strm;
createStatusStream(strm);
int height;
rect.top = 10;
rect.left = 10;
rect.right = 50 + tm.tmAveCharWidth;
height = pDC->DrawText(strm.str(), -1, &rect, DT_EXPANDTABS | DT_CALCRECT);
rect.bottom = height + 10;
pDC->DrawText(strm.str(), -1, &rect, DT_EXPANDTABS);
// Resize the scrollbars to fit the information it contains.
CSize size = CSize(rect.right+10, rect.bottom);
SetScrollSizes(MM_TEXT, size);
if (m_bDoResizeStatusView)
{
m_bDoResizeStatusView = FALSE;
ResizeStatusView(size);
}
// Once we call .str(), we must delete the allocated space.
delete strm.str();
return;
}

////////////////////////////////////
void createStatusStream()
{

```

```

// Insert a stream of characters in to the ostream passed in by
// the caller, which describes the status. The state argument
// indicates our current program state, which influences what
// information is included in the stream data.
// CDBView::createStatusStream(ostream& strm)
{
    CDBDoc* pDoc = GetDocument();
    int state = pDoc->GetState();
    PackedMsg *pMsg = pDoc->GetPackedMsg();
    strm << "\t\tSTATUS INFORMATION\n\n";
    switch (state)
    {
    case NO_IMAGE:
        // This case shouldn't come up - no menu access.
        strm << "No image has been loaded.";
        break;
    case IMAGE_LOADED:
        strm << "\tThe loaded image hasn't been signed or read.";
        break;
    case IMAGE_SIGNED:
    case IMAGE_SIGNED_AND_VERIFIED:
    case IMAGE_SIGNED_AND_SAVED:
        strm << "Signed Status\n\n";
        strm << "\tOriginal Text:\t\t" << pMsg->getAsciiMsg() << "\n\n";
        strm << "\tMessage Length:\t\t" << pMsg->GetMsgLength() << "\n\n";
        strm << "\tGain Setting:\t\t" << pDoc->GetSignerParams()->GetGain() << "\n\n";
        // strm << "\tGamma:\t\t" << pDoc->GetSignerParams()->GetGamma() << "\n\n";
        strm << "\tKey:\t\t" << pDoc->GetSignerParams()->GetKey() << "\n\n";
        strm << "\tBump Size:\t\t" << pDoc->GetSignerParams()->GetBumpSize() << "\n\n";
        strm << "\tDetail Gain:\t\t" << pDoc->GetSignerParams()->GetDetailScale() << "\n\n";
        strm << "\tChecksum:\t\t" << (unsigned) pMsg->GetSignerChecksum() << "\n\n";
        strm.fill(' ');
        // Change fill character for timestamps
        t = pDoc->GetSignerParams()->GetTimestamp();
        strm << "\tTime of Signing:\t\t";
        // Disable the 4270 warning. This is a bug in Microsoft's iomanip.h.
        // Without this, the setw() io manipulator causes a warning.
        #pragma warning(disable:4270)
        strm << setw(2) << t.GetHour() << ':' <<
            << setw(2) << t.GetMinute() << ':' <<
            << setw(2) << t.GetSecond() << '.' <<
            << setw(2) << t.GetMonth() << '/' <<
            << setw(2) << t.GetDay() << '/' <<
            << setw(2) << t.GetYear() - 1900;
        strm << "\n\n";
        strm.fill(' ');
        // Reset fill character to default.
        // Put the warning level back to the default.
        #pragma warning(default:4270)
    if (state == IMAGE_SIGNED_AND_SAVED)
        strm << "\tSigned image saved as:\t" << pDoc->GetFilename() << "\n\n";
    if (state == IMAGE_SIGNED_AND_VERIFIED)
    {
        strm << "Reader Status\n\n";
        strm << "\tRecognized Text:\t\t" << pMsg->getRecoveredAsciiMsg() << "\n\n";
        // Remove references to "super reader" for now
        //if (pDoc->GetSignerParams()->getSuperReaderFlag())
        //    strm << "\tAlternative Reader:\t\t" << "On" << "\n\n";
        //else
        //    strm << "\tAlternative Reader:\t\t" << "Off" << "\n\n";
        // Adjust the floating point precision of the stream.
        strm.setf(ios::fixed, ios::floatfield);
        strm.precision(2);
        strm << "\tBit Success Rate (%) \t\t" << pMsg->GetPercentCorrect() << "\n\n";
        // Print crude metric.
        strm.precision(4);
    }
}

strm << "\tBit Estimator Std. Dev.:\t" << pDoc->GetMetric() << "\n\n";
// Print range.
strm << "\tEmbedded Checksum Read:\t" << (unsigned) pMsg->GetReaderChecksum()
    << "\n\n";
strm << "\tChecksum Calculated:\t" << (unsigned) pMsg->GetComputedReaderChecksum()
    << "\n\n";
break;
default:
    break;
}
// Add a null terminator (DrawText needs it).
strm << '\0';
}
// Resizes the status view frame window. The goal is to not
// move the upper left corner, and to not exceed the bounds of
// the MDI main frame window on the right or left borders.
//void CDBView::ResizeStatusView(CSize status_size)
{
    const int bar_height = 27; // An empirically derived kludge
    CRect main_frame_rect, view_win_rect, view_client_rect;

```


SNOWTMP.CPP

```

////////////////////
// My experimental member function which
// builds a snow image in place.
//
//
void CDbDoc::MakeSnow(void)
{
    int cxDIB, cyDIB;
    long num_pixels, num_colors;
    LPSTR lpDIB, lpSnowyDIB; // Pointer to BITMAPINFOHEADER
    LPBITMAPINFOHEADER lpDIBHdr, lpSnowyDIBHdr;
    LPSTR lpDIBBits; // Pointer to DIB bits
    char __huge *src_data, *dest_data; // Huge ptrs for copying the image.

    HDIB hUnsignedDIB = GetHDIB();
    if (hUnsignedDIB == NULL)
        return;

    // Create space for the unsigned DIB for the snowy image.
    m_hSnowyDIB = (HDIB) ::GlobalAlloc(GMEM_MOVEABLE | GMEM_ZEROINIT, m_dwTotalDIBSize);
    if (m_hSnowyDIB == 0)
        return;

    // Here I follow the similar code in PaintDIB() of dibapi.cpp
    lpDIB = (LPSTR) ::GlobalLock((HGLOBAL) hUnsignedDIB);
    lpSnowyDIB = (LPSTR) ::GlobalLock((HGLOBAL) m_hSnowyDIB);

    src_data = (char __huge *) lpDIB;
    dest_data = (char __huge *) lpSnowyDIB;

    // Copy the BITMAPINFOHEADER, palette, and actual image byte data.
    for (image_byte = 0; image_byte < m_dwTotalDIBSize; image_byte++)
    {
        dest_data++ = src_data++;
    }

    lpDIBHdr = (LPBITMAPINFOHEADER) lpDIB; // Ptr to bitmap info hdr at start of dib.

    // Get ptr to the snowy dib header space, and copy header into it.
    lpSnowyDIBHdr = (LPBITMAPINFOHEADER) lpSnowyDIB;
    *lpSnowyDIBHdr = *lpDIBHdr;

    lpDIBBits = ::PindDIBBits(lpDIB);
    lpSnowyDIBBits = ::PindDIBBits(lpSnowyDIB);
    src_data = (char __huge *) lpDIBBits;
    dest_data = (char __huge *) lpSnowyDIBBits;

    // Copy the actual image byte data.
    for (image_byte = 0; image_byte < m_dwTotalDIBSize; image_byte++)
    {
        dest_data++ = src_data++;
    }

    cxDIB = (int) ::DIBWidth(lpDIB); // X size of DIB
    cyDIB = (int) ::DIBHeight(lpDIB); // Y size of DIB
    num_pixels = (long) cxDIB * cyDIB;
    num_colors = ::DIBNumColors(lpDIB);
    if (lpDIBHdr->biCompression != 0)
    {
        TRACE("Can't cope with compressed image (compression = %d)\n", lpDIBHdr->biCompression);
        ::GlobalUnlock((HGLOBAL) hUnsignedDIB);
        return;
    }
    TRACE("width = %d, height = %d, num_pixels = %ld\n", cxDIB, cyDIB, num_pixels);
    TRACE("num_colors = %d\n", num_colors);
    if (num_colors == 0 || num_colors == 16)
    {
        TRACE("At this time, only build snowy image for 8 bit images\n");
        ::GlobalUnlock((HGLOBAL) hUnsignedDIB);
        return;
    }
}

}

if (num_colors == 256)
{
    CxKey cxKey(1, (BITMAPINFO *) lpDIBHdr, lpDIBBits);
}

::GlobalUnlock((HGLOBAL) hUnsignedDIB);
}

// This is a part of the Microsoft Foundation Classes C++ library.
// Copyright (C) 1992 Microsoft Corporation
// All rights reserved.
//
// This source code is only intended as a supplement to the
// Microsoft Foundation Classes Reference and Microsoft
// QuickHelp and/or WinHelp documentation provided with the library.
// See these sources for detailed information regarding the
// Microsoft Foundation Classes product.

// stdafx.cpp : source file that includes just the standard includes
// stdafx.pch will be the pre-compiled header
// stdafx.obj will contain the pre-compiled type information
#include "stdafx.h"

// This is a part of the Microsoft Foundation Classes C++ library.
// Copyright (C) 1992 Microsoft Corporation
// All rights reserved.
//
// This source code is only intended as a supplement to the
// Microsoft Foundation Classes Reference and Microsoft
// QuickHelp and/or WinHelp documentation provided with the library.
// See these sources for detailed information regarding the
// Microsoft Foundation Classes product.

// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
#include <afxwin.h> // MFC core and standard components

```